universität
wien

D I S S E R T A T I O N

Titel der Dissertation

An Infrastructure for Context-Dependent RDF Data Replication on Mobile Devices

Verfasser

Dipl.-Inf.(FH) Stefan Zander, MSc, M.I.T.

Angestrebter akademischer Grad

Doktor der technischen Wissenschaften (Dr. techn.)

Wien, 2012

Studienkennzahl lt. Studienblatt:	A 786 175
Dissertationsgebiet lt. Studienblatt:	Wirtschaftsinformatik
Betreuer:	Univ.-Prof. Dipl.-Ing. Dr. Wolfgang Klas

Declaration of Authorship

I, Stefan Zander, declare that this thesis with the title, 'An Infrastructure for Context-Dependent RDF Data Replication on Mobile Devices' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

"The creation of something new is not accomplished by the intellect but by the play instinct acting from inner necessity. The creative mind plays with the objects it loves."

Carl Gustav Jung (1875 - 1961) Swiss psychologist

Abstract

This work describes an infrastructure for the selective RDF data replication to mobile devices while considering current and future information needs of mobile users and the different contexts they are operating in. It presents a novel approach in synthesizing context-aware computing concepts with semantic technologies and distributed transaction management concepts for intelligently assisting mobile users while enhancing mobile information seeking behavior and increasing the precision of mobile information retrieval processes.

Despite the huge potential of a proactive, context-dependent replication of RDF data, such data can not be efficiently processed on mobile devices due to (i) technical limitations and network-related constraints, (ii) missing processing and management capabilities of ontology-based description frameworks, (iii) the inability of traditional data replication strategies to adapt to changing user information needs and to consider technical, environmental, and infrastructural restrictions of mobile operating systems, and (iv) the dynamic and emergent nature of context, which requires flexible and extensible description frameworks that allow for elaborating on the semantics of contextual constellations as well as on the relationships that exist between them. As a consequence, existing approaches suffer from the deployment of proprietary data formats, server-dependent application infrastructures, and the inability to share and exchange contextual information across system borders. Moreover, results of recently conducted studies reveal that mobile users find their information needs inadequately addressed, where a large share can be attributed as context or context-relevant.

Although progress has been made in applying semantic technologies, concepts, and languages to the domain of context-aware computing, a synthesis of those fields for the proactive provision of RDF data replicas on mobile devices remains an open research issue. This work discusses possible fields where context-aware computing can be enhanced using technologies, languages, and concepts from the Semantic Web and contains a comparative study about the performance of current mobile RDF frameworks in replication-specific tasks. The main contribution of this thesis is a formal description of an abstract model that allows for an efficient acquisition, representation, management, and processing of contextual information while taking into account the peculiarities and operating environments of mobile information systems. It is complemented by a formal specification of a concurrently operating transaction-based processing model that considers completeness and consistency requirements on data and process level. We demonstrate the practicability of the presented approach trough a prototypical implementation of context and data providers that satisfy typical information needs of a mobile knowledge worker. As a consequence, dependencies to external systems are reduced and users are equipped with relevant information that adheres to their information needs anywhere and at any time, independent of any network-related constraints. Since context-relevant data are processed directly on a mobile device, security and privacy issues are preserved.

Zusammenfassung

Der im Rahmen dieser Arbeit vorgestellte Ansatz beschreibt die Erstellung einer technischen Infrastruktur, die selektiv RDF-Daten in Abhängigkeit der Informationsbedürfnisse und den unterschiedlichen Kontexten mobiler Nutzer auf ein mobiles Endgerät repliziert und diese somit in intelligenter Art und Weise unterstützt. Eine Zusammenführung kontextspezifischer Konzepte und semantischer Technologien stellt einen wesentlichen Bestandteil zur Verbesserung der mobilen Informationssuche dar und erhöht gleichzeitig die Präzision mobiler Informationsgewinnungsprozesse.

Trotz des vorhandenen Potentials einer proaktiven, kontextabhängigen Replizierung von RDF-Daten, gestaltet sich die Verarbeitung auf mobilen Endgeräten schwierig. Die Gründe dafür liegen in den technischen und netzwerkspezifischen Beschränkungen, in der fehlenden Verarbeitungs- und Verwaltungsfunktionalität von ontologiebasierten Beschreibungsverfahren sowie in der Unzulänglichkeit bestehender Replikationsansätze, sich an verändernde Informationsbedürfnisse sowie an unterschiedliche technische, umgebungsspezifische und infrastrukturbezogene Eigenheiten anzupassen. Verstärkt wird diese Problematik durch das Fehlen ausdrucksstarker Beschreibungsverfahren zur Repräsentation kontextspezifischer Daten. Existierende Ansätze leiden dementsprechend unter der Verwendung proprietärer Datenformate, dem Einsatz serverabhängiger Applikationsinfrastrukturen sowie dem Unvermögen, kontextspezifische Daten auszutauschen. Dies äußert sich in Studien, welche die Berücksichtigung der Informationsbedürfnisse mobiler Nutzer als unzureichend einstuft und einen Großteil der benötigten Informationen als kontextrelevant auszeichnet. Obgleich Fortschritte bei der Adaption von semantischen Technologien und Beschreibungsverfahren zur kontextabhängigen Verarbeitung zu erkennen sind, bleibt eine auf semantische Technologien basierende, proaktive Replizierung von RDF-Daten auf mobile Endgeräte ein offenes Forschungsfeld.

Die vorliegende Arbeit diskutiert Möglichkeiten zur Erweiterung der mobilen, kontextspezifischen Datenverarbeitung durch semantische Technologien und beinhaltet eine vergleichende Studie zur Leistungsfähigkeit aktueller mobiler RDF-Frameworks. Kernpunkt ist die formale Beschreibung eines abstrakten Modells zur effizienten Akquise, Repräsentation, Verwaltung und Verarbeitung von Kontextinformationen unter Berücksichtigung der technischen Gegebenheiten mobiler Informationssysteme. Ergänzt wird es durch die formale Spezifikation eines nebenläufigen, transaktionsbasierten Verarbeitungsmodells, welches Vollständigkeits- und Konsistenzbedingungen auf Daten- und Prozessebene berücksichtigt. Der praktische Nutzen des vorliegenden Ansatzes wird anhand typischer Informationsbedürfnisse eines Wissensarbeiters demonstriert. Der Ansatz reduziert Abhängigkeiten zu externen Systemen und ermöglicht Nutzern, unabhängig von zeitlichen, örtlichen und netzwerkspezifischen Gegebenheiten, auf die für sie relevanten Daten zuzugreifen und diese zu verarbeiten. Durch die lokale Verarbeitung kontextbezogener Daten wird sowohl die Privatssphäre des Nutzers gewahrt als auch sicherheitsrelevanten Aspekten Rechnung getragen.

Acknowledgements

First and foremost, I would like to thank my supervisor Univ.-Prof. Dr. Wolfgang Klas for his guidance and valuable support throughout the preparation of this work and in particular for his input and feedback on the formal model. Despite his positions as both chair of the research group and dean of the faculty of computer science, he always had an open ear for my matters. I also would like to thank my second supervisor Univ.-Prof. DDr. Gerald Quirchmayr for his helpful and constructive contributions during the doctoral seminars that have helped a lot in directing research efforts of this work.

Special thanks are due to my colleague and friend Dr. Bernhard Schandl or his encouragement and the many fruitful discussions we had – and with whom I authored a number of papers. Working with you Bernhard was both pleasure and true inspiration. Moreover, I want to express my gratitude to my present and former colleagues Maia, Gerhard, Peter, Christian, Ela, Bernhard, Chris, and Wolfgang as well as Niko, Wolfgang, Stefan, and Robert for their support and cooperation – it is and was a pleasure to work with you. A honorable mention goes to the two master students Doris Braunöder and Christina Ochsenhofer for implementing parts of the evaluation framework and for conducting the performance evaluation. Many thanks also deserve our two administrators Jan Stankovsky and Peter Kindermann for their hardware and software support as well as the members of our secretary's office for their administrative support.

Above all, I wish to express my sincere gratitude to my beloved family, friends, and Melanie for all their kind support and the sacrifices made throughout this work.

Contents

Declaration of Authorship **iii**

Abstract **v**

Zusammenfassung **vii**

Acknowledgements **ix**

List of Figures **xv**

List of Tables **xvii**

1 Introduction **1**
1.1 Mobile Computing 1
1.1.1 Mobility and Mobile Information Needs 2
1.1.2 Context and Context Awareness 4
1.1.3 Context and Context Awareness in Mobile Information Systems 5
1.2 Motivating Example 8
1.3 Problem Description 9
1.4 Contributions 10
1.5 Overview of this Thesis 14

2 Background **17**
2.1 Definition and Overview 17
2.2 Definition and Utilization of Context in Different Domains 20
2.2.1 Pervasive and Ubiquitous Computing 20
2.2.2 Artificial Intelligence (AI) 21
2.3 Positivist and Epistemological View on Context 22
2.4 Context and Context Awareness in Information Systems 23
2.4.1 Context Models 24
2.4.2 Classification Frameworks for Contextual Information 27
2.4.3 Reference Architectures for Context Acquisition, Management, and Processing 30
2.4.4 Key elements of Context Processing and Management Frameworks 39
2.4.5 Summary and Discussion 44
2.5 Problems of Context-aware Computing 45
2.6 The Semantic Web 47
2.7 Representing Contextual Information using the Resource Description Framework 49

2.7.1 Representing Contextual Aspects . . . 52
2.7.2 Identifying Contextual Information . . . 53
2.7.3 Representing Context Property Values . . . 53
2.7.4 Using Structured Properties for Representing Contextual Information . . . 55
2.8 Semantic Web-enhanced Context-aware Computing . . . 56
2.9 Discussion . . . 59
2.10 Summary . . . 60

3 Related Work and State of the Art 61
3.1 Introduction . . . 61
3.2 Mobile Data Replication . . . 61
3.3 Semantic Web Frameworks for Mobile Platforms . . . 62
3.3.1 Mobile XML Parsers . . . 65
3.3.2 Mobile RDF Frameworks . . . 66
3.3.3 Query and Persistence Frameworks . . . 68
3.3.4 Discussion and Summary . . . 69
3.4 Analysis and Review of Related Projects . . . 72
3.4.1 Related Projects and Applications . . . 75
3.4.2 Analysis . . . 83
3.4.3 Summary . . . 87
3.5 Conclusion . . . 87

4 Approach 89
4.1 Requirements and Design Considerations . . . 90
4.2 Conceptual Context Acquisition and Data Replication Workflow . . . 94
4.3 Formal Model . . . 98
4.3.1 Symbols and Relations . . . 99
4.3.2 Context Model . . . 102
4.3.3 Context Provider . . . 103
4.3.4 Orchestration Trees . . . 109
4.3.5 Formal Definition of the Orchestration Logic . . . 112
4.3.6 Compounded Context Acquisition Model . . . 114
4.3.7 Context Acquisition Workflow . . . 115
4.3.8 Context Configuration . . . 116
4.3.9 Context Description . . . 117
4.3.10 Data Providers . . . 118
4.4 Formal Model of the Orchestration Process . . . 119
4.4.1 Data Description Ontology . . . 121
4.4.2 Computing Compatibility Metrics between Context Providers . . . 124
4.4.3 Building Orchestration Trees . . . 133
4.5 A Transaction-based Processing Model for Context Acquisition Workflows . . . 136
4.5.1 Preliminaries . . . 137
4.5.2 Definition of Transactions . . . 139
4.5.3 Events and Event Histories . . . 143
4.5.4 Dependencies between Transactions . . . 145
4.5.5 Processing Context Acquisition Workflows . . . 149
4.6 Conceptual Architecture . . . 155
4.6.1 Concepts and Features . . . 155
4.6.2 Components . . . 157
4.7 Discussion and Summary . . . 164

5 Implementation and Case Study 171
5.1 Development Platform . . . 172

5.2 Implementation of Context and Data Providers 174
5.3 Case Study 179
5.3.1 Context Acquisition 181
5.3.2 Data Provisioning 182
5.4 Summary 184

6 Evaluation of the Processing Efficiency of RDF Data Replicas 187
6.1 Test Environment 188
6.2 Test Setup 189
6.2.1 Test data 190
6.2.2 Preparation of Test Data 190
6.2.3 Recording of Benchmark Results 191
6.3 Results 193
6.3.1 Parsing RDF Data Replicas 193
6.3.2 Serialization and Storage of RDF Data Replicas 201
6.3.3 Adding Data to RDF Data Replicas 210
6.3.4 Removing Data from RDF Data Replicas 222
6.3.5 Retrieving Elements from RDF Data Replicas 228
6.3.6 Constructing In-memory RDF Graphs 234
6.4 Discussion and Summary 236

7 Conclusion and Future Work 239
7.1 Conclusion 239
7.2 Future Work and Possible Application Fields 241

A Detailed Performance Statistics of the Replication Benchmarks 245

Bibliography 265

List of Figures

2.1 Roles and influence of context awareness in human-computer interaction with respect to effectuation, representation, and interpretation . . . 29
2.2 Layered reference architecture for context frameworks[1] . . . 31
2.3 Reference architecture of context-aware computing systems for ubiquitous environments[2] . . . 33
2.4 Crisp (a) and fuzzy (b) quantization methods for calculating context features from low-level context data[3] . . . 42
2.5 Example of low-level to high-level context-aggregation using Bayesian classifications[4] 43
2.6 Example of a situation ontology that classifies situations as business and private[5] 44
2.7 Example RDF Graph . . . 48
2.8 Value space, lexical space, and lexical-to-value mapping for the XML Schema data type `xsd:boolean` and their typed literal definition . . . 55
2.9 Use of `rdf:value` property for representing the values of a temperature sensor . 55

3.1 Summary of the evaluated mobile XML parsers, RDF frameworks, and query and persistence frameworks (summarized as *'Infrastructure Frameworks'*) . . . 71
3.2 The DBpedia Mobile client application depicting Semantic Web resources about POIs located around the user's current location on a map (left picture) as well as descriptions of the Brandenburg Gate retrieved from DBpedia and Revyu (right picture)[6] . . . 76
3.3 The mSpace Mobile client application showing cinemas located in the user's immediate vicinity (left picture) together with movie and actor information (center and right picture)[7] . . . 77
3.4 An overview of the IYOUIT system consisting of two screenshots of the client application (left side) as well as screenshots depicting different views of the community portal (right side)[8] . . . 79
3.5 The ContextTorrent system architecture and its core components[9] . . . 81
3.6 An example of NanoXML's functions for accessing OWL and RDF document elements[10] . . . 82
3.7 Summary of the evaluated Semantic Web projects . . . 84

4.1 Conceptual architecture of the components involved in a data replication process 95
4.2 Example of an adjacency tableau created from the elements $r \in R_p$ for a context provider $p \in P$ . . . 114
4.3 Structure of the data description ontology represented in EBNF . . . 123
4.4 Structural composition of elements constituting the data description ontology . . 124
4.5 Exemplary data description for a complementary context provider for extracting contact data from calendar entries . . . 125
4.6 Example of a compatibility matrix $\mathbf{M}^C$ and the corresponding orchestration matrix $\mathbf{M}^O$ . . . 133
4.7 Example of an adjacency tableau and corresponding orchestration matrix $\mathbf{M}^O$ for eight context providers $p_{1,\ldots,3} \in P$ and $c_{1,\ldots,5} \in C$ . . . 135
4.8 Orchestration trees o_{p_1}, o_{p_2}, and o_{p_3} derived from the orchestration matrix $\mathbf{M}^O$ depicted in Figure 4.7 . . . 136

4.9 Correspondence between the relations $r \in R_p$ constituting the structure of an orchestration tree $o_p \in O$ and the corresponding transactions $t \in T_p$ 141
4.10 Conceptual architecture of the proposed context-dependent RDF data replication framework 158

5.1 Conceptual class infrastructure for context and data providers 175
5.2 Code snippet of `GPSContextProvider`, converting location data into an RDF-based context model 177
5.3 Code snippet of the `DBpediaLocationDataProvider`, querying DBpedia for data about location resources 180
5.4 Geographical coordinates as returned by the `GPSContextProvider` (Turtle notation) 181
5.5 Context description as returned by the `GeonamesContextProvider` (Turtle notation) 182
5.6 Context retrieved from the user's calendar by `GoogleCalendarContextProvider` 182
5.7 Aggregated context models that constitute a context configuration instance . . . 183
5.8 An example SPARQL query produced by the `DBpediaLocationDataProvider` . 183

6.1 An excerpt of the DBpedia test data set used for the performance evaluation . . 191
6.2 Parsing performance of the Androjena framework depending on the serialization format 195
6.3 Parsing performance per device using the μJena framework 197
6.4 Parsing performance per device using the Mobile RDF framework 198
6.5 Parsing performance compartmentalized by device and serialization format . . . 200
6.6 Serialization and storage performance of the Androjena framework separated by serialization format 203
6.7 Serialization and storage performance using the μJena framework 205
6.8 Serialization and storage performance using the Mobile RDF framework 206
6.9 Serialization and storage performance compartmentalized by device and serialization format 207
6.10 File sizes in bytes of RDF data replicas stored on the local file system depending on the serialization formats supported by involved RDF frameworks 209
6.11 Performance of insertion operations on included devices using the Androjena framework 212
6.12 Performance of insertion operations on included devices using the μJena framework 214
6.13 Performance of insertion operations on included devices using the Mobile RDF framework 216
6.14 Performance per framework for adding data sets of specific size to RDF data replicas on the HTC G1 218
6.15 Performance per framework for adding data sets of specific size to RDF data replicas on the Motorola Milestone 219
6.16 Performance per framework for adding data sets of specific size to RDF data replicas on the Samsung Galaxy S I9000 220
6.17 Performance per framework for adding data sets of specific size to RDF data replicas on the Dell Streak 221
6.18 Performance of removal operations using the Androjena framework 224
6.19 Performance of removal operations using the μJena framework 226
6.20 Example of the query method signatures and result sets implemented in the Androjena and μJena frameworks 229
6.21 Example of the query method signature and result set implemented in the Mobile RDF API 229
6.22 Performance of retrieval operations using the Androjena framework 230
6.23 Performance of retrieval operations using the μJena framework 232
6.24 Performance of retrieval operations using the Mobile RDF framework 233
6.25 Construction of in-memory RDF graphs using the Androjena, μJena, and Mobile RDF frameworks 235

List of Tables

4.1 Symbols used in the formal model . . . 100
4.2 Relations existing between selected elements of the formal model . . . 101
4.3 List of the generic states $q \in Q$ of a context provider $cp \in CP$ . . . 107
4.4 Core classes of the data description ontology . . . 121
4.5 Domain and range values of the data description ontology properties . . . 122
4.6 Symbols and descriptions of the inverted term indices . . . 126
4.7 Symbols used in the transactional processing model . . . 140

6.1 Overview of the Android Devices' Specification . . . 189
6.2 File sizes in bytes of locally stored RDF data replicas depending on RDF framework and serialization format . . . 208

A.1 Detailed results of parsing RDF data replicas on the HTC G1 . . . 246
A.2 Detailed results of parsing RDF data replicas on the Motorola Milestone . . . 246
A.3 Detailed results of parsing RDF data replicas on the Samsung Galaxy S I9000 . . . 247
A.4 Detailed results of parsing RDF data replicas on the Dell Streak 5 . . . 247
A.5 Detailed results of storing RDF data replicas on the HTC G1 . . . 248
A.6 Detailed results of storing RDF data replicas on the Motorola Milestone . . . 249
A.7 Detailed results of storing RDF data replicas on the Samsung Galaxy S I9000 . . . 250
A.8 Detailed results of storing RDF data replicas on the Dell Streak 5 . . . 251
A.9 Detailed results of adding data to RDF data replicas using the Androjena framework on the HTC G1 . . . 252
A.10 Detailed results of adding data to RDF data replicas using the Androjena framework on the Motorola Milestone . . . 252
A.11 Detailed results of adding data to RDF data replicas using the Androjena framework on the Samsung Galaxy S I9000 . . . 253
A.12 Detailed results of adding data to RDF data replicas using the Androjena framework on the Dell Streak 5 . . . 253
A.13 Detailed results of adding data to RDF data replicas using the μJena framework on the HTC G1 . . . 254
A.14 Detailed results of adding data to RDF data replicas using the μJena framework on the Motorola Milestone . . . 254
A.15 Detailed results of adding data to RDF data replicas using the μJena framework on the Samsung Galaxy S I9000 . . . 255
A.16 Detailed results of adding data to RDF data replicas using the μJena framework on the Dell Streak 5 . . . 255
A.17 Detailed results of adding data to RDF data replicas using the Mobile RDF framework on the HTC G1 . . . 256
A.18 Detailed results of adding data to RDF data replicas using the Mobile RDF framework on the Motorola Milestone . . . 256
A.19 Detailed results of adding data to RDF data replicas using the Mobile RDF framework on the Samsung Galaxy S I9000 . . . 257
A.20 Detailed results of adding data to RDF data replicas using the Mobile RDF framework on the Dell Streak 5 . . . 257

A.21 Detailed results of removing data from RDF data replicas using the Androjena framework on the HTC G1 258
A.22 Detailed results of removing data from RDF data replicas using the Androjena framework on the Motorola Milestone 258
A.23 Detailed results of removing data from RDF data replicas using the Androjena framework on the Samsung Galaxy S I9000 259
A.24 Detailed results of removing data from RDF data replicas using the Androjena framework on the Samsung Galaxy S I9000 259
A.25 Detailed results of removing data from RDF data replicas using the μJena framework on the HTC G1 260
A.26 Detailed results of removing data from RDF data replicas using the μJena framework on the Motorola Milestone 260
A.27 Detailed results of removing data from RDF data replicas using the μJena framework on the Samsung Galaxy S I9000 261
A.28 Detailed results of removing data from RDF data replicas using the μJena framework on the Dell Streak 5 261
A.29 Detailed results of retrieving elements from RDF data replicas using the Androjena, μJena, and Mobile RDF frameworks 262
A.30 Detailed results of constructing in-memory RDF graphs using the Androjena, μJena, and Mobile RDF frameworks 263

Dedicated to my family and all the people who supported me throughout my life

Chapter 1

Introduction

"Mobility is perhaps the most important current market and technological trend in information and communication technology (ICT)"

Prof. Barbara Pernici (Politecnico di Milano)

Recent trends in information and communication technology indicate that mobile information systems are becoming increasingly prevalent today. They allow for accessing information resources and services independently of physical locations or temporal constraints. Mobile devices also became an integral part of our everyday lives for managing our personal information assets where more and more aspects are virtualized. This trend is accelerated by research towards creating ubiquitous and pervasive computing environments (cf. [AMR02]) as evident from recent deployments of near-field communication technology and communication networks (cf. [Mad08, YKIAL09, BJRGN10]). Traditional mass media like television or radio broadcast that were originally separated due to different technologies and separated communication channels are now becoming ubiquitously integrated and tailored towards mobile consumption. This technological convergence of traditionally separated communication channels together with recent technological advancements in terms of network technologies and protocols enable mobile users to consume a wide variety of services and resources and send information across mobile telephone networks regardless of any physical connection. New mobile infrastructures and more reliable network connections with a higher Quality of Service (QoS) will change the way how people access and consume information [Per06]. With such convergence of technology and network infrastructure, resources and services can be accessed from virtually *anywhere*, i.e., regardless of the current location or prevailing equipment, at *anytime*, and *anyhow* using multimodal communication and interaction channels. This form of mobility opens the door for new, more sophisticated types of mobile applications.

1.1 Mobile Computing

Mobile computing has revolutionized the way information systems are used and utilized [FZ94]. While early mobile devices had been designed as part of a superior computing infrastructure and require other components for being fully operational [FZ94], the current generation of mobile devices are self-contained autonomous computing devices. They offer continuous and unobstructed

access to online available information built on multiple versatile communication channels that could be established between communicating partners. This led to a paradigm shift from consuming information towards producing information and participating in so-called social communities and networks, where mobility adds another dimension: participation in such networks is no longer a matter of spatio-temporal and physical circumstances but possible almost anytime and anywhere.

The increasing advent of mobile devices also introduced a new computing paradigm that moves from *general-purpose* towards *task-specific computing* [SBwG98]. Mobile devices in general need to be increasingly aware of environmental, technical, and user-related changes since mobile interaction is rather spontaneous and situation-dependent compared to desktop-based interaction where the constituting environmental parameters remain relatively stable [MT07, ATH07]. One reason for making devices and systems context-aware is to refrain people from the "infelicitous" interference of technology [Eri02]. Context-aware computing is supposed to be a technological solution to solve or at least minimize that issue, especially in consideration of the ongoing technological advancements in pervasiveness, ubiquitousness, and mobility.

1.1.1 Mobility and Mobile Information Needs

Mobility not only influences the type of information we need but also how we access it as well as the tools and mechanisms that are at our disposal to process it. Understanding the information needs of mobile users and their interaction metaphors is crucial for improving mobile application development and the usability of mobile devices in general [SLGH08].

An extensive study conducted to identify mobile search pattern in search queries reveals that mobile search differs substantially from desktop search in terms of intra-query diversity[1], the effort needed for setting up a query, and the total amount of queries initiated in one session [KB06]. However, this study only focuses on how people use search engines rather than what information they are really seeking for while taking into account motivation and situational circumstances of users, i.e., *the context* in which a query was initiated. In contrast, Sohn et al. [SLGH08] conducted a qualitative diary study to analyze the information needs of moving people and how they address such needs, i.e., the strategies and methods they use to retrieve the required information by observing search behaviors.

It turns out that plain internet access is often not sufficient for adequately addressing information needs of mobile users since their situational contexts and current activities could not be addressed sufficiently (also see [CS08]). Other issues concern the impedimental interaction with the device while browsing for information, the extensive attention needed for interaction and information seeking tasks, and the lack of methodological knowledge on how to address a specific information need although users had the required resources (tools and permissions etc.) at their disposal [SLGH08]. As a consequence, sole internet access does not solve or contribute to the majority of mobile users' information needs.

Despite the technical limitations of mobile devices, there had been attempts (e.g. [McL02, XMS+05, KTCY09]) to convert desktop browsing interfaces to mobile screens. However, deploying desktop metaphors on mobile devices does not seem to be appropriate for mobile users since mobile applications and services need to be tailored to the specific needs and requirements

[1] The diversity of queries initiated in mobile setting is significantly lower compared to queries on desktop search. Additionally, query categorization reveals that context searching behavior is similar to desktop searching behavior although query exploration is significantly lower.

of mobile users [SLGH08]. Personal information needs often depend on the particular situation—the context—a user is currently in. Recent endeavors (e.g., Google GOOG-411[2], Google mobile[3], or Microsoft Live Search[4]) try to address these issues to provide mobile context-sensitive services, but still require substantial human involvement. In this respect, it is crucial to understand the types of information mobile users need as well as how these needs are addressed.

According to that study, 72 percent of information needs are attributed or related to context[5]. Context-aware computing can significantly support the task of information access and proactive opportunistic information delivery. Due to the fact that context is likely to change the needs of users and how they address it, a context-aware computing infrastructure should be incorporated into the essence of mobile devices [Teo08]. By analyzing the user-related activities, users can be supported with information related to their current tasks at hand in a proactive and opportunistic manner. Calendar items as a special form of timely orchestration of events might help in sorting and determining information that might become important on future events. However, an estimated amount of 58 percent of mobile information needs could be satisfied by requesting publicly available data sources [SLGH08] wherefore we consider Linked Data repositories (cf. [Biz09, BHBL09]) as one of the main sources for local data replication. The information hosted in Linked Data sources is potentially relevant for a wide range of application domains and in particular for mobile applications. In the following, we exemplify the benefits of replicating Linked data to a mobile device by means of three application scenarios:

- *Efficient Mobile Personal Information Management.* The quality of mobile personal information management can be significantly improved by augmenting the personal space of information with publicly available data that may originate from Linked Data sources: 58% of mobile information needs can be satisfied by publicly available data [SLGH08]. For instance, data from FOAF profiles can be directly integrated into the user's local address book, therefore keeping the user's contacts database up-to-date. Similarly, events that are of interest to the user can be directly imported into their calendars, if event data are available on the Web in structured form.

- *Location-based Services.* When users are traveling with their mobile devices, it is quite common for them to use location-based and social services to become informed about their current location. For instance, information from geographical services can be combined with semantically annotated news feeds and user reviews in order to form a unified, context-aware information stream. Therefore, *semantic mashups* bring additional user benefit that a single information source cannot provide.

- *Emergency Management.* Especially in crisis regions (e.g., after an earthquake), first-aiders cannot expect the local infrastructure (especially power supply and cellular network coverage) to operate normally. However, being equipped with the right data can be of significant importance to persons who are not familiar with the local circumstances. Therefore, it is crucial that in such situations relevant information (such as local points of interest, amenities, infrastructure, etc.) is proactively replicated to their mobile devices in order to allow aiders to perform their work more efficiently.

Additionally, 38 percent of users' information needs are of personal nature and can be solved by accessing personal information items; having an infrastructure that proactively retrieves and

[2]Google GOOG-411 service: `http://www.google.com/goog411/`
[3]Google mobile service: `http://www.google.com/mobile/`
[4]Microsoft bing: `http://www.discoverbing.com/mobile/`
[5]See [SLGH08] page 440 and Figure 5(a).

updates such items can help people saving time and effort. Future mobile systems should therefore take into account a user's context as well as her personal data items stored across multiple sources to better address mobile information needs [CS08, SLGH08].

1.1.2 Context and Context Awareness

Humans in general have five senses through which they gather information in form of bio-chemical processes that constitute the basis of our actions, believes, judgements, and how we conceive the surrounding environment and reality in general [KPL+04]. These collected or "sensed" information form the basis of our judgments and our way to conceive *our* reality. Based on the interpretation of signals collected by our senses, we are able to conceive a given situation and take appropriate actions. To simulate that behavior technically, devices are equipped with little microprocessor-driven sensors that allow for gathering specific aspects of the surrounding environment electronically and represent them as machine-processable raw data. By interpreting and aggregating such data, machines are able to assign "meaning" to those collected data streams and draw inferences to "understand'" a given situation and process it accordingly. Context and context awareness are thus thought of as the machine-equivalent to the human capability of judging a situation and taking appropriate actions [KPL+04]. Context is also of great importance regarding the selection processes of humans: context helps people to determine relevant aspects of the current situation thus preventing us from *information overload* that originates from environmental stimuli [MT07]. The central idea of context as discussed in the computer science literature is to *bind information processing and communication tasks to situational aspects* [SBwG98]. The most prominent example of context-dependent adaptation can be found in the implementation of context menus in graphical user interfaces, which adapt their menu items according to the selected element.

Context combines information processing and communication tasks with aspects of the surrounding environment and relates them to the personal concerns of individuals, their goals, tasks at hand etc. [Bon04]. One way to achieve such form of awareness about the surrounding environment and the different situations a mobile device is operated in, is by formalizing and processing contextual information. The central question in this respect is what is context, how can it be formalized and processed, and how can context awareness be formalized and utilized for mobile information systems [GSB02]. Our underlying assumption is, the more a device knows about its user, the environment, and the situation in which it is used, the better assistance it can provide in accessing relevant services, delivering relevant information, and contribute to an overall user experience (cf. [SBwG98, ATH07]). According to this assumption, the user should take a central role since their relationships to the surrounding environment often determine the relevance and interpretation of contextual aspects.

Context awareness in its simplest form describes a system's capability to conceive aspects of the physical and virtual environment it is operating in and dynamically adapt its behavior and internal decision making processes according to the computational analysis of such aspects [Dey00, FMGI06]. Context awareness can therefore be defined as a system's capability of using contextual data for providing relevant information and services with respect to the current situation of the user [Dey01]. A system can be denoted as "context-aware" when it is able to adapt its behavior to ongoing activities, as well as the operational environment where it is used in [ST94, ADB+99]. Context-aware applications and systems are able to react according to those changes in an intelligent and user-related manner [ATH07].

This can be accomplished by sensing and interpreting changing conditions, resources, and processes [Dey00, FMGI06]. Context awareness can also be thought of as the machine-equivalent to the human capability of judging a situation and taking appropriate actions [KPL+04]. From a technical viewpoint, context awareness refers to the accurate extraction, combination, and interpretation of contextual information, gathered from various multi-modal sensors [BC04] where its objective refers to the identification of the set of relevant features that describe and represent a given situation with the greatest possible accuracy [SBwG98]. The key aspect in this respect lies in answering the question which information sufficiently characterize the situation in which a user currently operates or at least helps in identifying it.

An important aspect in this respect is *relevance* ([Bon04]) that defines the meaningfulness of contextual information by measuring their relevance according to the user's activities and current tasks at hand. Assuming that every piece of information can be considered context-relevant (cf. [ADB+99]), we need to define a *filter* that distinguished between related and relevant-related contextual information as they directly influence the outcome of a system's responses and adaptation processes [Bon04, BCQ+07]. The main objective of context-aware computing therefore is to structure and select those information that can be defined as relevant to users and their tasks at hand and thus becoming context-relevant.

Attempts to categorize context and context-aware approaches suffer from the non-existent availability of a common and general understanding about context and context-awareness across domains [SBwG98]. Context in general is defined differently across communities where the focus of most approaches was put on the acquisition and processing of contextual information rather than finding a unified model and theory of context [Bon04]:

> *"Context is not any more a matter of knowledge representation or knowledge processing, but information about a concrete environment of a person, device, computer network."*

Early attempts in context-aware computing in particular suffer from the exclusive concentration on specific forms of contextual aspects (in most cases location) and context approximations that were derived from environmental aspects and provide either too low or too specific abstractions, or were designed for specific application domains [SBwG98]. However, this form of conception and treatment amplify the problems in unifying and consolidating the heterogeneous context definitions and models, which manifest themselves in a lack of generality, flexibility, and extensibility[6] [Bon04]. Recent approaches therefore claim for representing context using generic models augmented by languages and frameworks that allow for the explicit specification of meaning using semantic technologies. These will be introduced and discussed in the course of this thesis.

1.1.3 Context and Context Awareness in Mobile Information Systems

Context awareness should be incorporated into mobile information systems for the following reasons [ATH07]: (i) user contexts change more frequently due to mobile behavior, (ii) mobile devices are operated in highly dynamic environments with constantly changing user requirements, and (iii) mobile information needs are different from those of desktop users wherefore search capabilities[7] need to be augmented with contextual information to tailor result sets towards the user's current situation and information needs [SLGH08].

[6] These problems are discussed in Section 2.5.

[7] For instance query adaptation and query expansion

Throughout this work, we use the term 'mobile information system' according to the definition provided by [Per06]:

> **Definition 1.1** (Mobile Information System)**.** *A mobile information system is an information system that allows for accessing resources and services independently of spatial and temporal constraints by using end-user terminals typically based on wireless connections. In a mobile information system, resources and services can be accessed by a multitude of different devices through different communication channels.*

Within the domain of Mobile Computing, context is generally used for multiple purposes: (1) increasing the accuracy of the information retrieval process, (2) adaptation of device behavior according to the environment in which it operates, and (3) increasing the usability and interaction between a user and the device (cf. [SBwG98, KPL+04]). This is achieved by either filtering the flow of information (i) from the device to the user to decrease information overload as well as (ii) from the user to the device, where user-generated data is augmented with contextual information predominantly in an automated and transparent manner to add additional meaning. The underlying rationale is that mobile communication requires the user's explicit concentration and attention where context-assisted communication will introduce new interaction styles, techniques, and paradigms (e.g. transparent interaction [Abo99]) that in turn reduces the cognitive load and explicit interaction with the device (cf. [KPL+04]). Contextual-aware systems in general provide a "more natural and less obtrusive way of interaction" and contribute to an enhancement of the overall mobile usability and user experience [MT07]. Other research in the domain of mobile computing [KA04] has attempted to use context awareness for overcoming the technical limitation imposed by current mobile devices in terms of small screen sizes and limited interaction possibilities. Users often are confronted with multiple simultaneous activities and information channels, where context-aware computing promises to improve user interaction by reducing explicit user inputs and attention [KA04]. In this respect, context-awareness allows for a transition from traditional *explicit user-driven interaction design* to an *implicit context-driven interaction design* [SS00, GSB02]. This aspect is particularly relevant for mobile information systems since multiple simultaneously running applications are competing for the user's explicit attention. Context awareness can be therefore considered as a methodology for facilitating human computer interaction by lowering explicit cognitive load and user attention.

Deriving reliable information from multiple heterogeneous sources in uncertain and rapidly changing environments is mandatory for efficient context awareness in the domain of mobile computing [KMK+03]. The challenge in providing accurate and reliable context information lies in the detection and elimination of noisy, faulty, inaccurate, or—in worst cases—conflicting and contradictory data retrieved from heterogeneous sources. Especially in mobile and dynamic ad-hoc environments, context-relevant information can change rapidly and unpredictably, and evolve over time. Even if context-related data represent real-world contextual constellations precisely and accurate, users might find automated adaptation of application or device behavior irritating due to different conceptualizations of context [KMK+03].

The real challenges of context-aware computing thus lie in the accurate detection of user contexts under the assumption that they are not known a priori [ATH07] and to avoid and reduce the misinterpretation of context which might arise due to the complexity of context and its relative nature [WB05]. This requires a selection of an appropriate representation logic and a mechanism that allows to backtrack and resolve inappropriate actions resulting from context misinterpretations.

In information systems, context is often treated as a special resource that defines and is defined by the sensors being relevant to it, and which must be observed to detect context transitions. This implies that only one single context is active at any given point in time and that a context can be characterized technically by the number of active sensors and the properties they measure. Most context-aware approaches therefore used isolated subsets of context such as location, identity, or technical and physical characteristics whereas recent works (e.g. [KA04, EPR08, CRL+09, HDW09]) motivate to direct research into exploring the relationships between the different context elements and examining their impact on the efficiency on context-aware applications [KA04]. However, using Semantic Web languages and knowledge representation languages, such relationships can be made explicit and described in a structured and well-defined way using controlled vocabularies and ontologies that are based on formal logic (cf. Section 2.7).

However, recent research in integrating context awareness into the essence of mobile applications suffers from a "fundamental methodological weakness" and demands for solid design methodologies [RTA05]. This fact influences the development of effective context awareness methodologies and models, which are seen as a key requirement for delivering useful information related to the current user activities as well as their communication concerns. Another factor that exacerbates the deployment of context-aware applications in mobile systems can be attributed to the lack of conceptual models, methods, and tools that would promote and facilitate the design of context-aware mobile applications [DAS01]. Mobile devices typically operate in environments that are characterized by unstable states, unpredictable contextual conditions, dynamically changing locations of users etc. However, such conditions have a significant influence on the way how users interact with the device.

Context-aware computing for mobile and pervasive environments has not reached its full potential yet, where the semantic interoperability between context descriptions and context sources as well is still not handled satisfactorily. Hu et al. [HDW09] identified the lack of a *standardized infrastructure* for context acquisition, utilization, and semantic interoperability among context sources as the main reason why context-aware computing is only insufficiently supported in mobile and pervasive computing yet. The diversity and magnitude of contextual information, new interaction patters, mobile operation modalities, and the increasing growth of context-relevant data are additional reasons for the weak adoption of context-aware computing paradigms in mobile and ubiquitous computing.

One possible field where context-awareness can increase the quality of mobile information management is *proactive information provisioning* [SZ09a, SZ09b, ZS10, ZS11]. We can rightfully expect that an information system should be capable of providing information relevant to the user's current task. However, a system that is capable of doing this without the need for the user to explicitly issue search and retrieval operations can bring significant benefit, because often users are not capable of explicitly expressing their information needs [SLGH08]. This is especially true for mobile environments and their limited interaction possibilities. In the following section, we present an example scenario where a mobile user is supported by such a proactive information provisioning system.

1.2 Motivating Example

John is a representative of a medium-size software company. His tasks include to regularly contact potential customers in order to create awareness for his company's most recent products, to maintain relations with already existing customers in order to ensure their support and maintenance plans still work for them, and to represent the company at exhibitions, industry conferences, and relevant meetings.

His company maintains a customer relationship management software, a product database, a shared calendar system, a company-internal Wiki system as collaboration platform, and a shared file server to store all kinds of documents. Because of his job, John is often required to travel to abroad places. In consequence, he heavily relies on mobile infrastructure to get his work done. He has a powerful laptop, which he uses as his primary working device, as well as a mobile phone, which is used as his personal information management device.

When he is on travel, it is crucial for him to be equipped with all relevant information for his business meetings and other activities. However, he can never be sure to have online access to his company's network from wherever he goes, since certain limitations are in place: missing network coverage or security restrictions may prevent him from establishing a connection via the cellular network, and even if he manages to setup a connection, it may be slow and unreliable. For this reason, John often relies on local replicas of relevant data, which he stores on his laptop and (to a far lesser extent) on his mobile phone. However, because of the limited storage capacity of these devices and the necessary infrastructure, he cannot synchronize all data from all systems mentioned before, so he has to carefully select subsets of these data, which is a tedious and error-prone task.

This selection needs to be done before each trip, since he needs different information every time: this includes data about the (potential) customers he is going to meet (this includes organizations as well as persons), the locations and venues he is going to (including points of interest to visit in his spare time), latest information about the products he is trying to sell (which requires close cooperation with his company's product managers and development department), and data needed for his trip planning and administration (including timetables and travel accounting information).

A system that would be able to automatically select data for replication from a variety of systems would be highly desirable for John, since it would save him several hours of preparation time before each longer trip. Such a system could make use of a number of data sources, which provide valuable hints about which data could be of importance during his trip. First, John organizes all his upcoming appointments and travel plans in his digital calendar, which contains dates, locations, and participants of meetings. Additional information about people and organizations can be found in John's personal address book, as well as in the company's customer relationship management system. There, references to products that customers will use are mentioned; these refer to entries in the product database.

Additionally, the system could infer potential selling options from the interest topics that are stored for leads and potentials. Further, it can lookup information about locations and points of interests that John will visit from external public data sources, e.g., the Linked Open Data cloud. Further, it can find (via keyword lookups) articles from the company-internal Wiki system and the shared file server and replicate all these data to John's both mobile devices. For this purpose the system could rank each information item according to its assumed relevance, and replicate data according to the mobile devices' capacities.

During his trip, John will update and extend the replicated data with upcoming information (e.g., contact data and interests of new potential customers). Whenever his devices have sufficient network connection to his company network, the system should automatically synchronize his devices, upload changes, and update his local replicas according to possible changes in his context. After he has returned from the trip, all information is synchronized back to their origin systems, ensuring that no data are lost. If the system is able to track John's actual usage of replicated information during the trip, it can utilize this implicit feedback to adjust its relevance ranking algorithms, and therefore improve the selection for his next trip.

1.3 Problem Description

Given the different and contradictory views on context and context awareness, it is useful to consider the question of how applicable existing research in the field of context-aware computing is to the problem of mobile RDF data replication and Linked Data exploitation in general, and whether existing results in Semantic Web and context-aware computing research can be efficiently combined and deployed on mobile information systems to establish a context-sensitive infrastructure for the exploitation and replication of RDF data. This thesis contains a detailed review of the research literature of context and context awareness and elaborates on how context can be represented, processed, and stored using semantic technologies on a mobile information system. This analysis forms the basis for innovation in context-driven RDF data replication on mobile devices, uncovering useful concepts and techniques that can be applied to context-dependent mobile Semantic Web-based information systems, and offers improvements for the issues that current RDF frameworks for mobile platforms experience. The peculiarities and generic graph-based structure of the RDF data model render its processing especially on mobile systems cumbersome and inefficient (resource consuming) due to the lack of efficient frameworks and infrastructures for mobile RDF processing, management, and storage[8] (see [Zan09, SZ09a, ZS10] and Section 3.3).

The efficient acquisition and processing of distributed and heterogeneous RDF-based context descriptions highly depends on concepts and methodologies explored and researched by the database management systems (DBMS) community – especially in the domain of distributed transaction management (cf. [GR92]). Those findings need to be adapted to the peculiarities of mobile information systems to provide an infrastructure for the efficient context-dependent RDF data replication and storage for mobile information systems that is facilitated by technologies and languages from the Semantic Web.

Although progress has been made in applying context-aware computing research to Semantic Web research and vice versa (cf. [GMF04, Bon04, HMD05, EPR08, CRL+09]), the demonstration of a unified Semantic Web-based context management and processing architecture specifically designed for context-driven RDF data replication to mobile platforms still remains an open issue. Little work has been done in examining the creation of a context-dependent replication infrastructure for mobile systems to date [ZS11]. This places limitations on information systems targeted towards the mobile exploitation, utilization, and replication of RDF and Linked Data sources and also on a wider adoption of the *Web Of Data* [Biz09] in mobile information systems research.

[8] Other works (e.g. [MWL+08, Owe09]) investigated optimization strategies to increase RDF triple store performance; however, those issues become more obvious when RDF data is to be processed and stored on resource constraint systems such as mobile device or PDAs.

Although the replication and processing of RDF data on mobile devices offers a magnitude of significant advantages and new possibilities to mobile application development, such data cannot be easily deployed on and processed by mobile devices due to the following reasons:

- *Technical limitations:* despite the significant recent technical advancements in this field, mobile devices are still restricted in terms of memory capacity, computational performance, power supply, and heat generation. Therefore, potentially large Linked Data sets cannot be processed efficiently by mobile devices. Instead, algorithms are needed that facilitate an intelligent selection of potentially relevant data w.r.t. users information needs and tasks at hand.
- *Connectivity or network-related constraints*: network connectivity might be hindered by several factors, e.g., technically (no cellular radio coverage), economically (high transaction costs), or because of security restrictions (protocol restraints, blockade of several ports). These may render the usage of applications with a high degree of network traffic impossible.
- *Different application and operation models*: since mobile devices use different modalities in accessing information and are operated in different contexts, current tasks might be intermitted abruptly or moved to the background. Therefore different application models and operating system infrastructures have to be employed in order to deliver an appropriate user experience.
- *Missing RDF processing capabilities*: existing Semantic Web frameworks such as *Sesame*[9], *Virtuoso*[10], and *Jena*[11] are too heavy-weight to be efficiently deployed on mobile devices, while common data management frameworks for mobile systems provide only rudimentary RDF support.

These issues clearly indicate that mobile systems require a more sophisticated approach in making RDF and Linked Data available that is centered around user activities while considering the different contexts users may be operating in. To the best of our knowledge, there is no such system in existence yet, and we believe that this work will greatly inform and stimulate both the Semantic Web community as well as the mobile computing community in further combining both technologies for context-aware computing and contribute towards synthesizing research in those areas.

However, this work does not intend to solve the problems related to the absence of a general model of context and context awareness in mobile computing (cf. [SBwG98, RTA05]), but it helps in achieving a technological convergence towards open and flexible models for contextual information representation as well as processing and management architectures that better resemble the special characteristics and nature inextricably linked with the notions of context and context awareness.

1.4 Contributions

The central contribution of this thesis is a formal and conceptual specification of a Semantic Web-based context processing and management framework for mobile RDF data replication to facilitate the information needs of mobile users by proactively and transparently replicating

[9] Seasame: `http://www.openrdf.org`
[10] Virtuoso: `http://virtuoso.openlinksw.com`
[11] Jena: `http://jena.sourceforge.net`

data to the mobile devices that address current and future information needs. The proposed architecture supports the acquisition, aggregation, consolidation, and dissemination of contextual information acquired in a distributed fashion from heterogeneous context sources and allows for the exchange and reuse of context descriptions by using well-defined semantic vocabularies for describing contextual information. This work contributes to research in context-aware computing on mobile systems backed by semantic technologies, taking into account the peculiarities of mobile platforms and information systems and the lessons learned from the implementation of several aspects of the proposed architecture and the motivating example.

In the following, we outline the main contributions of this work and also present contributions that have been created in the context of this thesis but are published elsewhere:

- *A Semantic Web-based context processing and management framework for mobile RDF data replication*

  By replicating related data in a transparent and proactive manner to the mobile device, we can better address the current and future information needs of mobile users and foster the development of applications and services that utilize such information. The architecture was built on principles from graph theory and distributed transaction management for the acquisition, augmentation, and aggregation of context-relevant data gathered from a variety of different sources in a deterministic and well-defined manner while maintaining data and process consistency. It allows to combine independently acquired contextual information to derive additional, high-level context information that was not initially anticipated. We conceptually describe the set of constituting components that cover the entire context processing life cycle comprising context identification, acquisition, interpretation, aggregation, consolidation, reasoning, storage, and dissemination. This infrastructure can be deployed on mobile devices that contain a Java virtual machine and adapted to certain scenarios. It can serve as a basis for deploying more sophisticated services or application frameworks for instance for situational awareness (cf. [Geh08, LFWK08, SWB+08, CCMS10]) or for enhancing personal information management (cf. [BS04, Kel06]) through the connection to semantic desktop systems [HMD05, SBD05, FAS09]. The framework also contains a persistent storage mechanism for RDF data based on the μJena RDF framework that was extended with a graph-based storage implementation backed by the SQLite database provided by the Android platform. It exhibits two significant advantages compared to server-based approaches as it does not depend on the availability of an external system and all contextual data (which may include highly private information) are processed locally on the mobile device, which reduces security and privacy issues. Different aspects of our approach and the constituting architecture were published in several workshop and conference and papers as well as journal articles (e.g., [SZ09a, SZ09b, ZS10, ZS11, ZS12b]).

- *Formal model and conceptual architecture for the acquisition, aggregation, and consolidation of heterogeneous RDF-based context descriptions taking into account technical and conceptual peculiarities of mobile devices and mobile operating systems*

  The underlying formal model is based on the idea of cascading context acquisition components in orchestration graphs (cf. [AMR02]) that allow for a controlled and deterministic execution of acquisition workflows while maintaining data and process consistency and considering technical and conceptual peculiarities of mobile devices. From the analysis of existing works in related domains, we have identified a set of requirements that serve as a basis for the conceptual architecture in order to avoid the emergence of inaccurate, inconsistent, incomplete, or outdated contextual information. Due the dynamic and unpredictable character of mobile environments, compensation strategies have been included

in the conceptual model to deal with situations in which a context source is temporarily unavailable or malfunctioning. Aspects of the formal model were published as preliminary results in earlier works (e.g., [SZ10b, ZS12b]).

- *The specification and prototypical implementation of a formal graph-based orchestration model for context acquisition and aggregation based on a minimal, lightweight data description ontology*

  We have defined a lightweight ontology for describing the data a context acquisition component emits that serves as a basis for identifying compatibilities between context acquisition components and combining their acquisition workflows by cascading them in orchestration networks. The data description ontology allows for describing emitted data on different granularity levels that are considered by the orchestration algorithm. The compatibility among context acquisition components is computed by a matching algorithm based on configurable scores for correspondences on namespace, concept, and property levels. In addition, the orchestration algorithm considers RDFS semantics such as `rdfs:subClassOf` relationships. For instance, if one context provider emits `foaf:Person` instances and another context provider requires `foaf:Agent` instances as input data, the matching algorithm detects the compatibility between these differing concepts since `foaf:Person` is a subclass of `foaf:Agent` according to the FOAF ontology [BM07]. The building process is completely decoupled from the context framework where rebalancing the orchestration networks does not affect context acquisition tasks.

- *A comprehensive evaluation on quantitative and qualitative aspects of current mobile RDF frameworks*

  We have conduct a comprehensive evaluation regarding the features offered by currently available RDF frameworks for mobile devices. We focus on quantitative aspects regarding the parsing, storage, and processing performance as well as on qualitative aspects concerning ontology, language, query, and reasoning support, as well as quality of documentation, API robustness, licensing model etc. The analysis of quantitative aspects has been conducted on different classes of Android devices to obtain insights about the scalability behavior of mobile RDF frameworks with respect to available hardware resources and processing power. This analysis identifies potential areas where mobile RDF frameworks need to be improved and points out the weaknesses regarding the storage, processing, and query of large RDF data sets on mobile devices. To the best of our knowledge, no such comprehensive analysis exists to date whereas results from preliminary and less exhaustive studies were published in [ZS10, ZS11].

- *An evaluation of current frameworks and applications in the Semantic Web domain that synthesize context-aware computing with technologies and concepts from the Semantic Web*

  We have analyzed existing projects in the Semantic Web domain that already incorporate or are built on context-aware computing functionality and published preliminary results in [Zan09, ZS11, ZS12b]. Since existing works in the domain of ubiquitous and pervasive computing have been extensively surveyed (e.g. [ADB+99, CK00, RTA05, BDR07, Kja07, SB08a]), we exclusively concentrated on Semantic Web-related projects that aim to synthesize context-aware computing functionality with semantic technologies and languages and mobile information system aspects. For our analysis, we identified requirements that we consider relevant for classifying a system as context-aware. This gives readers an impression on the context-aware functionalities offered by current works and helps them classifying existing systems according to distinct requirements. To the best of our knowledge, no such analysis exists to date for mobile Semantic Web-based context-aware applications.

- *Showcase that demonstrate how the work of a knowledge worker can be improved through the proactive and transparent replication of publicly available RDF data*

  We demonstrate the feasibility of our approach through the prototypical implementation of a typical use case of a knowledge worker that is on a business trip including several meetings and he/she can not rely on a stable network connection. We implemented a number of context and data providers that acquire data related to the user's current position, combines them with personal information items extracted from the user's local address book and calendar, and replicate data about persons the user is likely to meet within the next days together with information about points of interest that are in her immediate vicinity to the mobile device. Details regarding the implementation of the case study were published in [ZS11].

- *A discussion of possible fields where Semantic Web technologies and concepts can enhance context-aware computing on mobile devices*

  Early approaches in context-aware computing suffered from a number of limitations and the usage of proprietary technologies and languages for processing and representing contextual information (e.g. [SBwG98, Dey01]). We could identify an number of similarities between the acquisition and processing of contextual information that has been acquired from distributed context sources and the Semantic Web, which was designed as an infrastructure to deal with distributed and heterogeneous data descriptions by reconciling syntactical, structural, and semantic heterogeneity and explicitly representing data semantics [BLHL01, WVV+01]. By applying those concepts and technologies to context processing and management, we are able to build an architecture that fosters interoperability and exchange among context descriptions in a semantically enriched way by using vocabularies being built on formal logic and well-defined semantics. We outline those areas of context-aware computing that benefit from the application of semantic technologies and demonstrate the usefulness for future context-aware computing projects. Main aspects of this issue are published [ZS12b].

- *A framework for the local storage and dissemination of replicated RDF data*

  To allow other applications to utilize replicated data, we have implemented an RDF-based content provider that offers access to local data replicas. The RDF content provider covers operations for accessing data replicas as well as business logic that ensures synchronization among replicas and modifications performed on them. It contains the projections between the graph-based representation of RDF data and the relational model exposed by the local SQLite database that serves as a mobile triple store. The RDF content provider can be easily utilized by other projects to implement a local RDF storage infrastructure. To provide a persistent storage mechanism for RDF data, we have extended μJena RDF framework with a storage implementation that is backed by the SQLite database provided by the Android platform and uses a *normalized triple table design* (cf. [AMMH09]) for storing RDF models. Moreover, μJena has been extended with lightweight support for *named graphs* [CBHS05]. Details of this work were published in [ZS12b].

- *Implementation of a mobile RDF browser for the visualization and navigation of local data replicas and external Linked Data sources*

  The implementation of a mobile RDF browser shows how locally replicated RDF data can be visualized in a user-friendly interface so that users can browse and navigate through data replicas. The RDF browser considers the semantics of selected RDFS [KC04] and OWL [PSHH04] vocabulary elements and adapts its rendering algorithms accordingly. Moreover, it extracts and renders multimedia objects contained in an RDF document.

This prototypical RDF browser implementation offers mobile users the possibility to experience the benefits of proactively replicated data without the burden of coping with technical or implementation details of the underlying infrastructure. It has been improved in a number of iterations and is to date the one and only available browser that offers such functionalities. Since it is implemented as an open-source project, it can be easily used and extended in other applications or projects. A detailed description of this work is to be published in [ZS12a].

1.5 Overview of this Thesis

In Chapter 2 we give a comprehensive overview of the notions of context and context awareness, elaborate on how they are used and understood across communities, outline their role and utilization in information systems, and discuss problems and limitations. We ascertain the two main streams, wherein context is either considered a *representational issue* reflecting environmental aspects, and as an *emergent phenomenon*[12] that is continuously re-negotiated between communicating partners and thus cannot be determined beforehand, especially not at the design time of a mobile system. We also give a brief introduction to the general idea of the Semantic Web, its main constituents and involving technologies, as well as its knowledge representation languages. The chapter concludes with a discussion of possible areas where context processing and management can be substantially enhanced by the deployment of Semantic Web technologies, leading to an approach that we denote as *Semantic Web-enhanced Context-aware Computing*.

Since currently available full-fledged RDF frameworks can not be deployed on mobile systems for several reasons (cf. [ZS10]), we comprehensively analyzed current RDF frameworks developed for mobile devices according to their RDF processing capabilities and provide an overview of their features in Chapter 3. This overview is followed by an analysis on the current state of the art of projects that emerged in the Semantic Web domain and aim to synthesize context-aware computing, technologies and languages from the Semantic Web, and mobile information system architectures. We analyze the capabilities and context-relevant features offered by those projects and provide an evaluation framework that helps readers classifying existing systems as context-aware.

This analysis serves as a basis for defining a number of requirements and design considerations for the proposed context-sensitive RDF data replication architecture, which is presented in Chapter 4. We formally describe the concepts and distinguishing features of the replication framework and give in-depth insights into the constituting components, formal models, and algorithms for mobile context acquisition, aggregation, consolidation, and dissemination. We also present a loosely-coupled context-acquisition model that resembles concepts from graph theory and distributed transaction management to maintain data and process consistency as well as a formal description of the graph-based orchestration model and an exposition of the data description ontology.

The prototypical manifestation of the formal model and conceptual system architecture is introduced and described in Chapter 5 together with a case study that is built upon the proof-of-concept prototype. In the first part, we introduce Android as a novel development platform for mobile applications. Distinctions to existing platforms are ascertained and some of the unique features of using Android as development platform for our approach are expounded. In the

[12] This perception reflects the dynamic character of context.

second part, we give and overview of the class structure of context acquisition and data replication components that allows 3rd-party developers to exclusively focus on the acquisition and replication logic rather than dealing with processing and management-related aspects. The feasibility of our approach is presented in the last part of Chapter 5 and illustrated through the implementation of a real-world use case introduced in Section 1.2. The use case serves as proof of concept of our chosen approach and demonstrates how the proposed framework can be used to proactively replicate RDF data on the mobile device to ensure that a knowledge worker is at any time equipped with data about people they are likely to meet in the near future, as well as information about people and points of interest that are based near the user's current position.

In order to evaluate the practical applicability of our approach, we present a comprehensive performance analysis about the processing efficiency of local RDF data replicas on modern mobile platforms using the proposed context-sensitive RDF data replication framework. In Chapter 6, we therefore analyze and discuss the runtime behavior of typical RDF processing operations applied to local RDF data replicas and evaluate whether data replicas can be processed on current mobile devices in reasonable time using available RDF frameworks. We used devices from different mobile marked segments representing different device classes in order to be able to make assertions about the scalability and memory dependency of our approach.

In Chapter 7, we conclude our work by summarizing the findings, which have been acquired throughout the preparation of this thesis and the implementation of the constituting components and discuss future research directions and potential application domains.

Chapter 2

Background

"Context is not simply the state of a predefined environment with a fixed set of interaction resources. It's part of a process of interacting with an ever-changing environment composed of reconfigurable, migratory, distributed, and multiscale resources."

Coutaz et al., *Context is Key* (2005)

This chapter is intended to provide fundamental background knowledge about technologies and concepts that are necessary for building a Semantic Web-based context management and processing infrastructure for mobile devices. We provide an introduction to the concepts of context and context awareness and ascertain the two main streams wherein context is either considered a representational issue reflecting environmental aspects, and as an emergent phenomenon that is continuously re-negotiated between communicating partners and thus cannot be determined beforehand, especially not at the design time of a mobile system. We show, how such dynamically evolving contexts can be represented and processed using technologies and concepts from the Semantic Web, while preserving its unpredictable and dynamic characteristics—a requirement neglected by most context-aware computing approaches [Teo08]. The section also gives a brief introduction to the general idea of the Semantic Web, its main constituents and involving technologies, as well as its most prominent knowledge representation languages. The section concludes with a discussion of possible areas where context processing and management can be substantially enhanced by the deployment of Semantic Web technologies, leading to an approach that we denote as *Semantic Web-enhanced Context-aware Computing*.

2.1 Definition and Overview

The notions of context and context awareness have been subject to controversial discussion and differing perception across communities. Several definitions have been proposed to context, depending on its actual usage as well as on the domain in which it was utilized. Context is originally used in different domains such as Natural Language Processing (NLP) for information extraction tasks, Human Computer Interaction (HCI) for facilitating the interaction between a human user and a device, and Artificial Intelligence (AI) for information integration purposes (cf. [SBwG98, GM03, GMF04]). The word context is derived from the Latin word *con*, which means 'together' or 'with', and *texere*, which is the present infinitive of *'texö'*, meaning 'to weave'

or 'intertwine'. A well-known and widely used definition of context has been proposed by Abowd and Dey [ADB+99] which define context as follows:

> *"Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves."*

This definition describes context as a set of situations and actions (cf. [ATH07]) that are subject of dynamic and frequent changes including the states of the involved entities (contextual information fragments). Context in general can be defined as *"everything that surrounds a user or device and gives meaning to something"* [Teo08]. In this respect, Coutaz et al. [CCDG05] identified three characteristics that are essential to the notion of context:

1. Context is not simply a state but part of a process that requires the involved system to behave correctly with respect to the process and situation in which it is used.
2. Context must be considered as a process and treated holistically where the utility and usability of a system depends on its means to represent contextual information and the degree of exchanged information rather than on the technical constitution and sophistication of the involved services and components.
3. A system's interaction model must be aligned to the user's individual view of the system, i.e., aligning and adapting a system's functions to the individual who uses the system as closely as possible. This requires an adaptation of the interaction space and an introduction of conceptual frameworks to reduce system and model mismatches.

The last fact is particularly important for mobile or ubiquitous computing since the interaction space there is ill-defined, unpredictable, and emerges opportunistically [CCDG05]. New interaction possibilities and techniques help in reducing such misinterpretations but require the deployment of well-defined, flexible, and interoperable underlying conceptual context frameworks.

In [ST94] context is referred to as an entity that comprises location, identity, objects, as well as changes that apply to those objects. Ryan et al. [RPM98] add the dimension of environment and time. More user-related definitions have been provided by [DAPW97] and [Dey98] in which context is defined relatively to the user's emotional state, their believes, intentions, and concerns. These definitions take into account people in the closest proximity of the user. Other definitions such as proposed in [HNBr97] and [Bro96] define context on more abstract levels and wider scale: context is considered as "*the aspects of current situations*" as well as "*elements of the user's environment that are known by a computer*" [BDR07].

Bolchini et al. [BCQ+07] define context as an abstract process rather than a profile that determines how humans interweave their experiences with the environment surrounding them to give meaning to something. Other authors (e.g., [Dou04, EPR08]) highlight that context is never universal in that it encompasses all information constituting a specific situation but always defined relative to a concrete situation. Coutaz et al. [CCDG05] characterize context as "*not simply a state of a predefined environment with a fixed set of interaction resources*" rather than part of an interaction process with the polymorphic and varied environment that is composed of "*reconfigurable, migratory, distributed, and multi-scale resources*".

However, due to the different proliferations on the notion of context, there has not been a clear consensus established on what context exactly is [Dou04], but there is a common agreement on what it is about: context is concerned with an evolving, structured, and shared information space that is designed and utilized to serve a particular purpose [CCDG05]. Anagnostopoulos et al. [ATH07] consider context as a means to solve the problem of generality due to efforts in formalizing the notion of context in logic-based systems, e.g., in first-order logic systems (cf. [ADB+99, GM03, GMF04]).

Contextual information, which are constituent parts of surround contexts, can be considered as any information that may be used for describing the situation the user or a device is currently operating in. Contextual information is gathered through a variety of different technologies and considered as computational abstractions over distinguishable virtual or real-word aspects that have a specific relationship to the current task at hand. Contextual information can be *static* (e.g., the date of a person's birthday) or *dynamic* (e.g., a person's location). Dynamic context information is usually captured *indirectly* using sensors. Additional characteristics for classifying contextual information have been proposed by [HIR02].

Basically, contextual information can be acquired *explicitly* where context-related information is manually specified by the user, or *implicitly* where context information is captured through communication using specific technologies such as sensor or network infrastructures, or by monitoring user behavior. The main focus of the framework proposed in this work lies on the implicit acquisition of contextual information, especially from physical sensors embedded in the device or from ubiquitous sensors located in the immediate vicinity, as well as *logical* and *software sensors* that extract context-relevant information from personal sources such as emails, calendars, or web services[1]. Two approaches can be distinguished for capturing environmental context: the first approach is based on the idea of *smart environments* [SBwG98] which provide an infrastructure for obtaining and disseminating contextual information by placing intelligence in the environment. The second approach focuses on *embedding sensors* in devices for acquiring contextual data related to the physical or virtual environment independently from the surrounding infrastructure or environment. In this respect, the challenge is to identify a set of relevant features used for capturing and describing a situation or parts of the environment sufficiently [BC04]. Deriving reliable information from multiple heterogeneous sources in uncertain and rapidly changing environments is mandatory for context awareness in the domain of mobile computing [KMK+03].

Consequently, two forms of context awareness can be found in information systems [GSB02]:

- *Direct awareness* shifts the process of context acquisition onto the device itself, usually by embodying sensors that obtain contextual information autonomously and independent from the surrounding environment or infrastructure; e.g., location ascertainment using the device-internal GPS sensor. This approach usually employes distinct methodologies and logics for performing context acquisition, reasoning, and computations for deriving higher-level contexts from raw-sensorial data.
- *Indirect awareness*, in contrast, captures contextual information by communicating with sensors or services via the surrounding environment or infrastructure. Context acquisition and processing is done exclusively by infrastructural services, where a mobile system acquires contextual information during communication. For instance, to capture the social context of a user, a mobile device may request data from social communities or portals; to track the user's location, a remote geocoding service (based on the user's IP address) may be employed.

[1] See Section 4.3.3 for a detailed discussion; Section 5.2 provides implementation details of logical sensors

Indirect awareness introduces the problem of *trustworthiness* of acquired context-relevant data, which is referred to as the *"dislocation of context-acquisition from context use"* [GSB02] since indirectly captured context might be *imperfect*, i.e., the acquired information might be inconsistent, inaccurate, incomplete, or out-dated [ATH07]. The objective of a context-aware system is to minimize such failures as far as possible by applying different methodologies and technologies such as reasoning, Quality of Context (QoC) (cf. [BS03, BTC06, PvHS07, SWvS07, MTD08]), fuzzy logic etc. to derive reliable indicators regarding the trustworthiness of acquired contexts.

Gellersen et al. [GSB02] propose a direct awareness approach for context acquisition by employing a multitude of diverse sensors since those are better suited to capture a larger set of diverse aspect of a situation compared to single, generic, and powerful sensors. The underlying rationale is that sensor diversity is key for a comprehensive situational recognition especially for devices with severe technical limitations, since valuable contextual information can be inferred with comparatively small computational effort [GSB02].

2.2 Definition and Utilization of Context in Different Domains

In the following, we elaborate on how context is used and understood in different domains. To illustrate the ambiguous character and utilization, we focus on the three most prominent domains in relation to context-aware computing, which are pervasive computing, ubiquitous computing, and artificial intelligence. We outline how context is perceived and defined in each domain and sketch its main purpose.

2.2.1 Pervasive and Ubiquitous Computing

Pervasiveness has first been introduced by Mark Weiser [Wei91] and describes the transparent and seamless integration of electronic sensors and devices—also called *appliances* [KPL+04]—in the every day life of humans, where the users' tasks are intended to play the most important role. The central idea of pervasive computing is to equip the surrounding physical environment with small electronic devices that act as an interconnected network of interactive computing devices [Wei91]. Those networks of small computing devices being seamlessly and unobtrusively integrated and dispersed across the surrounding physical environment operate autonomously and independent from any centralized control, which create a new form of interaction between people, computational devices, and technology [CCSC07]. Their idea is to offer a multitude of sophisticated services assisting humans in unobtrusive ways [Sat02]. Pervasive environments are characterized by frequent user changes, a high user mobility, as well as diversity of used communication and interaction devices and networks. Users in those environments prefer to use applications that are tailored towards their current tasks and situations where context-aware systems are able to adapt to those needs. Context awareness in pervasive environments is considered as the capability to support the development of personalized applications that operate in those environments [PvHS07].

In ubiquitous computing, context is generally used for augmenting information retrieval tasks in which queries are enriched with codified context information not being manually specified by the user as well as to dynamically reflect environmental changes and adapt a system's behavior and its responses to predefined usage patterns [Dou04]. Contextual information is predominantly conceived as a set of physical or user-related parameters where context has largely

been defined according to technical or computational aspects such as available memory and battery capacity, processor speed and processing load, display resolution, language, installed applications etc [DAS01]. Context awareness in the domain of ubiquitous computing refers to a scenario in which the primary functions of a service adapt to the current context of a physical object [REB+06]. Its purpose is to support humans in performing their activities by delivering services that dynamically adapt their behavior according to the situation being used [CCDG05]. Two processes are fundamental to ubiquitous computing: (1) recognition of users' goals and activities, and (2) adaptively aligning these goals and activities to available resources and services [GSS02]. Both processes are supported by context in such ways as context provides a structured and unified view of the surrounding environment in which the user operates [CCDG05].

2.2.2 Artificial Intelligence (AI)

In the Artificial Intelligence domain, context is mainly concerned with the representation and provenance of information. Its perception differs from other domains since the primary objective is to enhance existing reasoning techniques to enable contextualized, i.e, context-dependent reasoning using *Propositional Logic of Context* in which context is considered as a first class entity of a logical theory, as well as *MultiContext Systems* respectively *Local Models Semantics*, that is a partial and individual approximation of the world's theory – to name the most mature approaches [BCQ+07]. The notion of context in Artificial Intelligence is specifically targeted towards two main areas: (i) to determine the validity of information and (ii) to improve the accuracy of reasoning algorithms [EPR08].

Most approaches try to formalize the notion of context in order to ascertain and determine the trustworthiness and validity of a given assertion by reducing the inherently present ambiguities of real world information entities (cf. [Dey01, HIR02]. More recent approaches such as the one presented in [GMF04] apply Web principles and knowledge representation languages from the Semantic Web to express different kinds of contexts. The underlying triple-based data model of RDF is enhanced towards a quad-based model (quadruple) in which each assertion is augmented with provenance information (cf. [DFP+05, PFFC09]).

While there is a dissonance in perceiving context as "independent theories related to some particular knowledge field" and considering context as "concurrent viewpoints on the same information" mappings should be defined for transforming and sharing information within different contexts [EPR08]. This approach is useful if heterogeneous context descriptions of comparable real world facts or stimuli (e.g different location sensing techniques) need to be integrated into a coherent corpus of interrelated context information.

Other works such as [WVV+01] and [Noy04]) apply the notion of context to the problems of aggregating data from multiple heterogeneous sources since although identical (data) models and vocabularies are used, there exists differences and inconsistencies in the way terms are applied, or regarding the assumptions underlying a data model. Guha et al. [GMF04] for instance examine some of the problems that may occur in aggregating independently published data on the basis of Semantic Web technologies and propose a context-based mechanism to handle those inconsistencies and incompatibilities. Although the directed graph-based structure of RDF (see Section 2.6) has proven to be a rather simple but effective model for the integration of heterogenous data, "higher-level differences" often exacerbates a direct integration since such difficulties occur due to assumptions made by data publishers that are implicitly reflected in the design of the underlying data models [GMF04].

2.3 Positivist and Epistemological View on Context

The rationale of many definitions for context that emerged in the information systems discipline is that information available in the physical and electronic environment influences human-machine interaction, thus forming the context for such interactions [DAS01]. The technical or positivist school therefore treats context as a conceptualization of human action and their interaction with the system, where the notion of context is intended to incorporate a specific form of sensitiveness to interactive systems[2] that allows them to conceive their operational setting electronically and respond appropriately [Dou04]. As a consequence, context is considered as a set of features of the environment surrounding the user's tasks at hand and generic actions, which can be computationally captured, represented, and processed. Especially technical disciplines consider context a representational issue and concentrate on sensorial or static data such as location, time, identity etc. (cf. [SBwG98]) putting emphasize on its codification and representation [MT07, Teo08]. Following this argumentation, context is instance-independent, separable from user activities, and can be scoped in advance [Dou04]. This form of perceiving context has its root in information system since it adheres very well to existing software methodologies [Teo08] but is contradictory to the phenomenological view of context that is grounded on subjective and qualitative analysis. This view considers context and activity inextricably connected and fundamental in giving meaning to something [Dou04]:

> *"Rather than considering context to be information, [the phenomenological view] instead argues that contextuality is a relational property that holds between objects or activities. It is not simply the case that something is or is not context; rather, it may or may not be contextually relevant to some particular activity. [...] the scope of contextual features are defined dynamically."*

This consideration emphasizes the aspect of relevance as context as such can not be defined or determined in advance since its scope is dynamically defined and changes frequently and unpredictably. It should be considered an *occasioned property* to emphasize its dynamic, fluent, and relative character as context arises in the course of action [Dou04]:

> *"Context isn't just "there", but is actively produced, maintained and enacted in the course of the activity at hand. [...] Context isn't something that describes a setting; it's something that people do. It is an achievement, rather than an observation; an outcome, rather than a premise."*

The epistemological view considers context as an interactional feature inspired by "sociological investigations of real-world practices" [Dou04]. This view has been adopted by many context-aware approaches in interaction theory since context and its surrounding activity are *inseparable* and require a holistic treatment of both aspects including their relationships and consequences w.r.t the user's tasks at hand [WB05]. Context is inextricably linked to the process in which it is conceived, which renders the positivist and phenomenological view on context incompatible. As [WB05] points out:

> *"Action and context are inseparable and should be analyzed as a whole. This implies that context awareness should be examined in its relationship and consequences to the supported activity or actions."*

[2] One major disadvantage of traditional interactive systems is that they only insufficiently respond to the environments in which they operate [Dou04].

This view puts emphasis on the stringent and close binding between *content* and *context*, and claims for a holistic treatment of both areas. Therefore, context and content should be considered and processed together since context arises and is sustained by the activity itself [Dou04, WB05]. Context in its broadest sense is essentially about the ways in which actions can be rendered as meaningful according to a particular situation. Context plays an important role in determining the meaningfulness and appropriateness of actions and information where the central question is, how can the dynamic and emergent nature of context be formalized and transformed to a model reflecting these characteristics, followed by the question how to support the process by which context is manifested, defined, negotiated, and shared in mobile environments (cf. [Dou04]).

The social or phenomenological school in contrast tries to intertwine context with certain aspects of social science as well as the social settings of the users. Context emerges opportunistically (cf. [CCDG05]) and reflects a mutual understanding of the course of user activities and interaction with a system. Information becomes contextually relevant if it has a certain influence on the user's tasks (e.g. [DAS01]). Therefore, the user should hold a central position in context-aware computing.

As a consequence, the dynamic aspects of context are entirely neglected by technically oriented disciplines as context is a constitutional part of interaction processes that emerge opportunistically. Context should be rather considered as an emergent phenomenon or feature of interaction that is inextricably linked with user activities [WB05, Teo08] and continuously renegotiated between communicating partners [Dou04, CCDG05, MT07] wherefore a pre-determination of the relevance of contextual elements is impossible – especially at design time of a system [EPR08].

To cope with this dynamic and emergent nature of context, a context processing and management framework must facilitate the creation of flexible, extensible, and open context descriptions that are not restricted to a single static vocabulary or predefined schema. Static context descriptions are not able to deal with unknown context information at run time, but require links between different context vocabularies to be specified at design time [EPR08]. The ability to dynamically handle and integrate new types of context information into existing structures is therefore a fundamental requirement of a context framework where open and well-accepted vocabularies help in describing contextual information to guarantee their evolution and accurateness. In this respect, one can observe an analogy to the "real" Semantic Web which deals with providing infrastructure to process information in a distributed and heterogeneous manner.

2.4 Context and Context Awareness in Information Systems

The notion of context is mainly used in the information systems discipline for two reasons (cf. [Dou04]): (1) contextual information facilitates information integration processes by encoding provenance information and (2) contextual information is utilized for adopting systems to the environments in which they are used. In mobile computing, in contrast, context is used for (i) intelligent service provision, (ii) realizing adaptive user interfaces, and (iii) increasing the accuracy of information retrieval processes since context-aware information systems in general provide a "more natural and less obtrusive way of interaction" [MT07]. This is achieved by filtering the flow of information from the device to the user to decrease information overload as well as from the user to the device, where user-generated data is augmented with contextual information predominantly in an automated and transparent manner. A variety of research endeavors elaborated on the nature of context and proposed a multitude of different definitions

as well as—mostly taxonomical—classification schemes. A large share of classification schemes distinguish between *physical context*, that is, contextual information retrieved from physical or hardware sensors referring to the surrounding physical or virtual environment, and *logical context* that encompasses user-related information such as a user's goals, their tasks, emotional states etc. Such information is acquired by monitoring user interactions, specified by the users themselves, or inferred from physical contexts. Other works (e.g. [BD05b]) classify context as *meaningful* if it has a particular relationship to the user's current high-level goals, or *incidental* in case the user enters an unpredicted or unexpected situation that has no special relationship to their high-level goals.

2.4.1 Context Models

The efficient management of a large number of diverse contexts require a working organization structure in which relevant and present information has to be tailored to the physical, cognitive, and social constraints imposed by the surrounding contexts defining a situation [CX06]. A *context model* has been identified as a central component (cf. [WX06]) for an open context management framework since it provides an abstract representation of context-relevant aspects of the surrounding real-world environment in a structured and machine-processable way. A context model's task is to act as a bridge between the physical world and a context-aware system [Dey01] since it allows for a separation application logic and application behavior from context acquisition and processing components [KMS+05], hence facilitating context exchange and interoperability between context infrastructures [WX06] where *adaptation* and *extensibility* have been identified as most important features [PdBW+04]. A large number of formal and informal context models have been proposed so far. This indicates one of the main problems context-aware computing suffers from: there exists no standard model on context and context awareness.

A context model in general is used for the definition and storage of contextual data in a machine-representational form. Due to the diversity of contextual information, several categorization frameworks have been introduced to manage and comprehend such information systematically. It reflects different aspects of a context-aware system where the contextual information represented by a context model may range from unstructured raw sensorial data constituting low-level context to full-fledged logic theories about contextual constituents (e.g. based on situation ontologies such as proposed in [LMWK05, LFWK08]). The most prominent approaches have been summarized in [SP04] and [BDR07], ranging from simple key-value models to complex logic or ontology-based models. The use of concepts from knowledge representation languages allow for using symbolic structures for representing context on which machine-based reasoning techniques based on formal logics can be applied. Context models can be distinguished according to the data structures they use for the representation and exchange of contextual data (cf. [SP04]):

- *Key-Value models* represent the simplest data structure for representing contextual data in a machine-processable form. They were especially used in service frameworks for describing service capabilities by key-value pairs on which basis matching algorithms can be applied to support the service discovery process. Although key-value models provide a simple form of managing and representing contextual information, they lack extended capabilities for structuring contextual data and representing relationships among contextual entities.
- *Markup scheme models* usually employ a hierarchical data structure that contains markup tags and corresponding attributes. A wide variety of markup schemes rely on the Standard

Generic Markup Language (SGML) or on one of its most prominent descendants XML. Markup schemes are usually represented as *profiles*. A discussion of the variety of different context profiles can be found in [SP04].

- *Graphical models* host general purpose modeling frameworks such as the Unified Modeling Language (UML) or Object-Role Modeling (ORM). Due to their graph-based representation and their generic structure, they are appropriate means for representing contextual information. Some extensions have been proposed to general purpose graph-based modeling languages (e.g. [HIR02]) for expressing complex contextual constellations and relationships.
- *Object-oriented models* aim to exploit the advantages of the object-oriented paradigm in describing and modeling the dynamics inherently anchored in the nature of context (cf. Section 2.3). Object-oriented concepts such as abstraction, inheritance, or polymorphism are applied to representing contextual data where the details are encapsulated in specific components and well-defined interfaces are used for their controlled access.
- *Logic-based models* were primarily used for deducing or inferring new facts from given contextual constellations. Usually, logic-based system employ a set of rules for describing contextual conditions in a formal way. Such systems usually employ a high-degree of formality and tend to be more general than specific.
- *Ontology-based models* represent contextual information by means of the concepts and properties defined in an ontology. Ontologies incorporate machine-processable semantics that allows for explicitly describing contextual facts as well as their interrelations and have proven to be appropriate means for representing contextual constellations in information systems (cf. [HMD05, MT07, EPR08]).

A context ontology serves as a uniform representation of contextual information and enables a systematic management of context-relevant aspects. It should be separated from application logic [KMS+05] and facilitates application adaptation, automatic code generation, code mobility, and user interface adaptation [PdBW+04], which have been identified as the key issues in ambient intelligence and mobile computing [RF05]. A number of approaches (e.g. [KMS+05, REB+06]) use single ontologies for context information representation and for the transformation of raw, low-level context data into high-level context descriptions (e.g. [BKL+08a]). Some of these ontologies refer to the analogy of physical objects, i.e., their concepts refer to objects in the real world (the studied context). In addition, [FMGI06] suggest to augment context descriptions with additional meta data about the *context source* a context description was created from, the *quality* of the sensing mechanism, *context value metrics*, *time stamps*, and *relationship attributes* to increase their precision and to support processing and dissemination steps.

Rather than using a single context model (cf. [KMK+03]), different works (e.g. [WX06, BKL+08a]) recommend to use multiple context models dedicated to different aspects of a context-aware system such as a *core context model* for identifying those context elements that are contained in and are relevant for a context representation, an *infrastructure context model*, which allows for integrating context consumers and producers by abstracting over concrete technical implementations, and *application-specific context models* for describing application-relevant aspects[3].

A different approach is taken by [RSP07] who motivates the usage of a "more user-friendly approach" in representing contextual entities, since those entities are mainly constructed for electronic consumption through a natural formalization of contextual entities oriented on human language, which seems more appropriate and usable than a rigid formal specification. Therefore,

[3] A number of requirements for each of the three context model types can be found in [WX06].

a *term-index-model* is introduced that allows for a verbal description of contextual entities such as user preferences using natural language constructs. An evaluation of the chosen approach regarding performance, user-friendliness, scalability, and other criteria is subject to future work.

A number of approaches (e.g., [BC04, REB+06, BKL+08a, BKL08b]) treat and represent context as hierarchies to better handle the complexity of contextual constellations and enable efficient inferencing by reducing the amount of rules to be applied to those constellations to a subset of the system's rule base. This yields positive effects on rule development and adaptation processes [BC04] since the number of rules to evaluate for a given context can be reduced from n, representing the entire set of rules deployed in a system's rule base, to k, which represents the average number of rules that need to be computed for an active context[4]. Therefore, the order $O(n)$ of rule-based inferencing is reduced to $O(k)$ where $k < n$ by combining rule-based reasoning and context hierarchies (cf. [BC04]); a formalization of this approach is presented in [GSB08].

Other works applied concepts from *Activity Theory* [CX06, Teo08] or *Task Analysis* [KA04] to describe contextual information independent from existing application scenarios aiming towards a general model of context awareness in mobile computing. Activity Theory is a philosophical framework developed in the 1920s for conceptualizing human activities and identifies significant elements that have an impact on human behavior. It accounts for the tools and the social environment such tools are used in since both areas have a substantial impact on how activities and tasks are performed by humans [KA04]. Cai and Xue [CX06] therefore propose an *activity-oriented context model* based on the computational theory of *collaborative plans*[5] [GK96] that establish late bindings of contexts to ongoing activities with respect to their contribution in finishing a task successfully. The late binding of activities, contexts, and adaption strategies allows for a more accurate determination of relevant contexts and scheduling of appropriate actions. They propose a computational model of activities consisting of a "diverse set of contextual factors", that relate relevant contexts to ongoing activities, where a certain aspect of the environment becomes context-relevant when it defines (or helps to define) a *contextual state*, which allows for the prediction on how to consume relevant services and information.

An aspect-oriented approach to deal with the *"crosscutting"* nature of context and to increase the efficiency in handling the different kinds of contexts and their relationships is proposed by [CCSC07] and [MFC07]. Quantitative results indicate an increased efficiency (positive effect) compared to conventional object-oriented programming techniques in terms of coupling, cohesion, and complexity [MFC07]. However, we did not follow this approach any further since aspect-oriented programming is not supported by our target development platform and would render a proof-of-concept implementation of our framework difficult if not impossible.

Several ontologies have been proposed for context modeling such as the *Context Ontology Language (CoOL)* from which the *Aspect-Scale Context Model* emerged [SLPF03], the *Context Broker Architecture (CoBrA)* [CFJ03], or the generic ontology proposed in [PdBW+04] for Ambient Intelligence. Other context modeling vocabularies such as *CC/PP* [Kis07b] revealed to be too restrictive for describing complex contextual constellations and models [IRRH03]. However, there is often a trade-off to be made between general purpose ontologies such as *DOLCE*[6], *SUMO*[7] [NP01], or *CYC* [LPS86] and specific domain dependent ontologies since workable

[4]Supposing that only one context is active at any given point in time (cf. [BC04]).

[5]A collaborative plan is a timely representation of activity-related tasks and actions, including associated constraints and contextual conditions. Every task involved in an activity is associated with a collaborative plan of activities, each of which having a certain relevance metric to determine its relevance towards successfully finishing a specific task.

[6]Laboratory for Applied Ontology - DOLCE: `http://www.loa-cnr.it/DOLCE.html`

[7]SUMO: `http://www.ontologyportal.org/`

context ontologies in general require a more precise description of exploitable context information [EPR08].

Ontologies are the preferred choice by most recent approaches to create and represent context descriptions as they allow to establish a common understanding of contextual semantics and relationships between context fragments that can be exchanged and communicated among communicating partners [FMGI06]. Ontologies further allow for performing complex context reasoning on contextual concepts to deduce additional, not explicitly asserted information. Context modeling thus focuses on improving context information accurateness, detecting and establishing relations among contextual entities, and reflecting contextual changes within a context model, which is denoted as *context model evolution*.

However, no "silver bullet" in context modeling has been proposed yet, since existing approaches merely focus on the facilitation of specific context sub problems and lack a deep understanding of the fundamental context problem, which is essential for the development of appropriate context models [BCQ+07]. A context model in general should be represented using a formalism (representation language) that allows to reason about contextual facts in polynomial time of complexity [ATH07]. It should be developed according to its application domain[8] [GSB02, EPR08] and focus on specific context sub-problems since the generality of a context model in terms of expressiveness and powerfulness is inversely proportional to its practical applicability and usability [BCQ+07]. In general, a lightweight vocabulary seems to be the most appropriate solution for modeling context in mobile systems [MT07].

2.4.2 Classification Frameworks for Contextual Information

For understanding the different types of context-aware systems and its architectures, it is crucial to understand the different types of context and their classifications [Kja07]. Categorizing context information allows for deducing or inferring new facts from given context descriptions and enable a more extensive assessment of a given situation [DAS01]. In the following paragraphs, we give an overview of the different classification schemes proposed for classifying contextual information. For further information, the reader is referred to the respective works.

A common approach for distinguishing contextual information is to classify context instances, that is, concrete manifestations or representations of contextual information or context descriptions according to *contextual dimensions* [BDR07], where a number of categorization frameworks have been proposed: Prekop and Burnett [PB03] distinguish between *external* and *internal* (also called *tacit*) contexts, that is, contexts focusing on supporting the user's cognitive activities. A *classification matrix* is proposed by Öztürk et al. [OA98] that adds the aspects of *relevance* and *independence* to external and internal dimensions. Chen and Kotz [CK00] use the contextual dimensions *influence* and *relevance* to classify context as *active* or *passive* according to the influence it has on application behavior and adaptation processes; a context is active when it directly influences an application's behavior, whereas passive contexts do not have a significant direct influence on an application.

Hofer et al. [HSP+03] and other authors divide context into *physical context* for representing concrete environmental aspects and instances, and *logical context* for complementing physical contexts with additional semantic information. Dey et al. [Dey01, DAS01] propose the three groups *places*, *people*, and *things* for classifying contextual entities, where various attributes can

[8] According to [GSB02] useful context must always be bound to a specific application domain where the usefulness of acquired contexts is always defined according to the application domain in which it is used.

be applied for further describing them. Those attributes itself can be classified into *identity*, i.e., the unique identifier each entity has, *location*, i.e., the physical or virtual location of an entity, *status*, i.e., additional properties that represent the implicit and explicit status of an entity, and *time* [BDR07].

In addition to physical context that reflects relevant attributes of the surrounding physical environment, Cai and Xue [CX06] differentiate between *computing or technical context* referring to the technical properties and capabilities of the involved device and network infrastructure, *human or social factors*, i.e., context that arises in the course of communication or human affairs, and *time*, which has considerable influence on people's behavior and helps to understand present and future actions [KA04]. However, works from related domains such as the Semantic Desktop (e.g. [HDM05, HMD05]) propose similar classification schemes for context, which distinguish between *personal*, *computational*, and *knowledge-based* contexts and emphasize the interactions among them.

Computational or technical context encompasses aspects such as network connectivity, installed applications, and device characteristics. Such information can be represented using specific vocabularies that exhibit concepts and properties for representing technical aspects of a system. For instance the *Composite Capability/Preference Profiles (CC/PP) vocabulary* [Kis07a], which is a W3C Recommendation, defines elements for representing device capabilities and user preferences. Version 2.0 of this specification is built upon the latest version of RDF using the concept of URIs for uniquely describing specific device functions and characteristics. It includes the function of RDF dereferencing [Lew07, BCH07] that allows for referring to subgraphs that are hosted on other machines or servers.

Krogostie et al. [KLO+04] identified four contextual dimensions that are relevant for context-aware computing in mobile information systems. The *spatial* dimension describes the movement of persons or objects in space as well as their interdependence among each other. A special form is the *spatiotemporal* dimension that adds the aspect of time and describes how things move along a time axis, which allows for deducing additional information such as direction, tracking, or speed. Ambient characteristics are referred to by the *environment* dimension that describe aspects related to the physical of virtual environment such as temperature, luminosity etc. The *personal* dimension refers to user characteristics such as physiological states (e.g pulse, blood pressure, weight collect via body sensors), mental states (e.g., mood, stress level, expertise), the current activities (e.g. goals, achievements, information needs), and social relationships [Per06, MT07].

Personal and social context are often used synonymously to refer to information related to the personal preferences of individuals including their social relationships and social networks. Such information substantially influence the way tasks or activities are performed while not being specific to it [HMD05]. Personal context is treated as the superordinate concept for social context and comprises individual preferences as well as personal resources that are relevant for a specific task. The authors in [PdBW+04] suggest to differentiate between *user profiles* which they classify as static, and *user preferences* that are always related to a situation and thus being dynamic. Mihalic and Tscheligi [MT07] emphasize that the user's social environment significantly influence the way mobile technology is used and claim for an interdisciplinary research approach to understand and elaborate on the interdependence between social context and technology[9]. A part of social context is *interactional context* [Dou04] that is based on the types of relationships between individuals. Such relationships can be described using specific vocabularies such as the FOAF vocabulary [BM07].

[9]During the act of communication, people modify their social relationships which in turn influence how mobile devices are used [MT07].

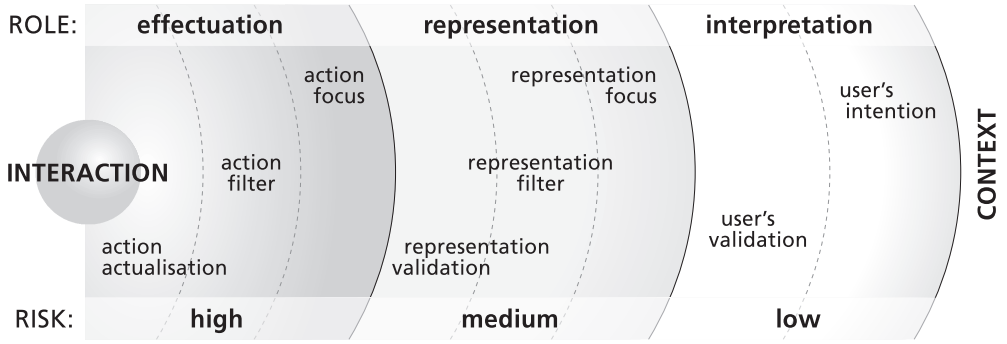


FIGURE 2.1: Roles and influence of context awareness in human-computer interaction with respect to effectuation, representation, and interpretation[10]

A classification framework for interactional context is proposed by [WB05] in which context awareness is classified into three high-level spheres according to its influence on interaction processed in human-computer interaction. The classification scheme describes the roles context can take in the interaction process and distinguishes between *interpretation*, *representation*, and *effectuation* (cf. Figure 2.1)

Within the *Interpretation Sphere* context is used for achieving an understanding of an activity's intentional meaning that results from the interpretation of and reasoning on low-level contextual data and allows for interpreting a user's intentional actions as well as their legitimization to perform a specific action. The *Representation Sphere* is responsible for the representational adaptation of contextual data to the user in order to reduce the cognitive load associated with conceiving the interfaces being part of the human-computer interaction process. This phase consists of three sub-phases: (i) selecting the representation focus and directing user's attention within the representation space, (ii) reducing information amount by providing only pertinent and hiding irrelevant information, and (iii) provide a valid representation of context information. The *Effectuation Sphere* is concerned with selecting and executing the most appropriate actions according to the contextual data acquired by applying sophisticated reasoning and machine learning techniques. This is done by suggesting the most appropriate actions (action focus), reducing the action space (action filter), and executing actions (reacting) on the user's behalf (action actualization). The *Representation* and *Effectuation Sphere* seize on Weiser's idea of transparency in which "interfaces disappear in the background" and computational tasks are hidden from the user [Wei91].

Kjaer [Kja07] classifies context according to whether it is *externally* or *internally* defined to an information system and argues that internally defined contexts are neglected wherefore they are not included in the context models of most context-aware computing approaches. Internal contexts represent information about the underlying computing infrastructure and the operating system such as the available disk space etc.

The theoretical framework developed by [RTA05] proposes four contextual dimensions that are complementary but interact with each other since contextual dimensions are not isolated in the real world [MT07]. *System context* addresses the entire set of interconnected devices and applications constituting a specific system as a whole from a technological, i.e., hardware and software-related perspective. *Infrastructure context* describes how the devices and applications

[10]Taken and adapted from [WB05], page 2.

being part of a system are connected among each other; such contexts are important for making explicit assertions about the validity of information, which might change when people and device move along the time-space continuum. *Domain context* refers to the social, physical, and mental concerns of users in terms of the information they require and information needs they have as well as "the specific situated interaction taking place in a domain" [RTA05]; the objective is to guide the user through a system's interaction space. *Physical context* encloses the physical properties of the surrounding environments and elaborates on how such properties are related to the entire system, for instance the current position of a device measured through a location sensor.

In [DAS01] context is categorized according to the four dimensions *identity* (i.e, assigning a unique identifier to each contextual entity or element to ensure its uniqueness within a certain namespace), *status* (i.e, intrinsic characteristics of a sensed or captured entity comprising technical, computational, personal or social, and environmental aspects, such as the temperature of a place represented by its geographical coordinates), *location* (i.e., geographical information in a three-dimensional space as well as additional information such as acceleration, elevation, or orientation etc.), and *time* (i.e., contextual information that helps in determining and characterizing a situation).

The taxonomy developed by [FMGI06] is based on the work of [ST94] and differentiates between *user domain*, *system domain*, and *physical domain*. User domain context can be further divided into *subjective* and *objective user context* where the first refers to the user's personality and psychological attributes such as their mood and feelings, the latter itself consists of *personal information*, *physiological* and *conditional information*, and *agenda-related information*. The system domain is concerned with technical capabilities and the technical specification of the operating device in terms of available hardware and software resources as well as installed applications. The physical domain comprises all information related to the physical, virtual, logical, and technical environment in which a device and its user operates. It is further subdivided into *physical geography*, *physical conditions*, and *timing information*. For a further discussion on the context taxonomy, the reader is referred to [FMGI06], page 37.

In summary, location-related information as part of the physical dimension is unarguably the most prominent type of contextual information, which receives the greatest attention in both mobile as well as pervasive and ubiquitous computing [GSB02]. It has been the subject of many research endeavors (e.g., [HKS06, PRS06, BD05a]) where powerful hybrid models have been proposed that do not only contain the physical coordinates of a particular location but also employ a combination of geometric, arithmetic, and symbolic models for representing logical relations such as `locatedIn`, `containedIn`, `northOf`, `nearBy` etc [DRD+00]. Early models, in contrast, are built on static structures consisting of fixed sets of attributes for representing location-based information and geographical entities. Newer hybrid approaches employ instance and schema-based models that expose abstract and logic modeling primitives. For instance, the location model proposed in [BD05a] employs a graph-based representation scheme comparable to the one proposed by RDF and RDFS.

2.4.3 Reference Architectures for Context Acquisition, Management, and Processing

The diversity of contextual information is also reflected in the variety of architectures proposed for context acquisition, management, and processing, which differ in their technical capabilities, acquisition techniques, context representations, and reasoning capabilities etc. The main

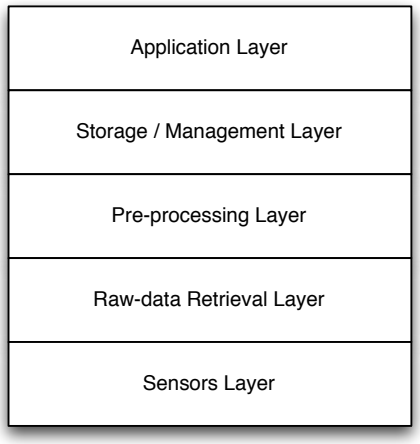


FIGURE 2.2: Layered reference architecture for context frameworks[12]

problems lie in the multitude of different information types and the weak standardization effort [RSP07]:

> *"Developers of context management infrastructures are not only challenged by the overwhelming amount of data that is to be stored in the system's database, they must also standardize a plethora of information types. Yet, albeit this approach appears to be straightforward, it comes at the cost of extensive standardisation."*

Despite the wide variety of different context management and processing architectures, a common architecture is identifiable [BDR07][11]. Figure 2.2 displays this conceptual architecture which consists of five separate functional layers:

- *Sensors layer*: The sensors layer is the bottommost layer and comprises the set of sensors that a context framework exploits. Such sensors can be classified into three groups [IS03]: (i) *physical* or *hardware sensors* for gathering physical context information, (ii) *virtual sensors* that capture data form software applications and service by observing user interaction and user behavior, and (iii) *logical sensors* that are responsible for deriving higher-level context information usually by augmenting physically or virtually acquired contexts with additional information from repositories or other data sources.
- *Raw data retrieval layer*: This layer is concerned with the retrieval of raw sensorial data usually by encapsulating driver logic or sensor APIs in dedicated wrapper components (cf. Widgets [DAS01]). It offers common interfaces that encapsulate specific drivers or sensor APIs thus making them exploitable for upper layer components. This layer exposes query-functionality that abstracts from low-level raw sensorial data and provides higher-level representations and functions.
- *Pre-processing layer*: In case sensorial data is too coarse [BDR07] or need to be clustered for further processing, this is handled by the pre-processing layer. It also provides appropriate abstractions of contextual data whose representation is too technical (e.g. a bit-wise representation of contextual data). Interpretation, aggregation, and reasoning tasks are also handled by this layer together with quantization algorithms for aligning raw-sensorial data with the elements of a framework's context model.

[11] See [BDR07] for a more detailed discussion about the different layers.
[12] Adapted from [BDR07].

- *Storage and Management layer*: This layer serves as a context repository as it organizes contextual information and offers interfaces to client applications. Components in this layer expose different operation models where contextual information can be either requested in a polling-based style where the client applications issues requests to a context server in regular time (*synchronous*) or via a subscription mechanism where clients are notified by the context server whenever new contexts are available (*asynchronous*). Due to the dynamic and unpredictable nature of context (cf. [Dou04, CCDG05, Teo08]) the use of an asynchronous communication style (cf. [XYCS02]) is suggested [BDR07].
- *Application layer*: The application layer is the uppermost layer and hosts the client applications that request and process contextual information from a context repository or a framework respectively. Contextual information is usually consumed by applications in this layer in order to adopt their actions with respect to user-related tasks. One popular example is increasing the background luminosity level in case the user enters an outdoor area.

In a similar way, Dey et al. [DAS01] identified five distinct conceptual components a context framework must consist of. *Context acquisition components* are used for collecting and gathering contextual data, that is, raw sensorial data or context-relevant data from other sources such as web services or ubiquitous devices located in the immediate vicinity. *Interpretation components* are responsible for deriving useful and workable information from sensed data by building abstractions and deducing qualitative high-level assertions. Interpreters are often used in conjunction with *context aggregation components*, so called *aggregators* to combine multiple context data fragments and allow for a uniform access by applications. *Context services* [DAS01] or *accentuators* [BC04] respond and react according to acquired and aggregated contextual information provided by a context framework. This can either result in an adaption of device or application behavior, which includes an adaption of its user interfaces towards the current needs of the user, or the adoption of requests to remote services (context-aware interaction and service delivery). *Context discovery components* scan the environment for devices and sensors that can be used as context sources and provide information regarding their accessibility and utilization, e.g., the used communication protocols, network addresses etc. These five types of components correlate to the layers proposed by [BDR07] in terms of their functional responsibilities.

A different layer-based reference architecture particularly for ubiquitous environments has been proposed by [CCDG05]. It consists of four horizontal and three vertical layers and addresses both technical as well as functional abstractions for a reference architecture for context-aware computing. In the following, we briefly sketch the constituting elements of the proposed general-purpose architecture, which is depicted in Figure 2.3.

On the bottommost level, the *sensing layer*, a contextual aspect or fact is represented as a set of (plain) numerical values sensed from physical sensors embedded in the device or located in the environment. This layer is responsible for the acquisition and transformation of raw contextual data into so called *numeric observables*. To derive useful and valuable meaning from numeric observables, these must be interpreted, aggregated, or clustered (cf. [GSB02]) as necessary so that algorithms and transformation heuristics could be applied. This is done on the *perception layer*, which is ideally independent from the sensing layer and the underlying sensing technology respectively. It maps raw context data values to symbolic values (add reference to quantization) to provide high-level contexts (cf. [BKL+08a, LFWK08]). The *situation and context identification layer* elaborates on the current situation of the user by analyzing the

[13] Adapted from [CCDG05], page 52.

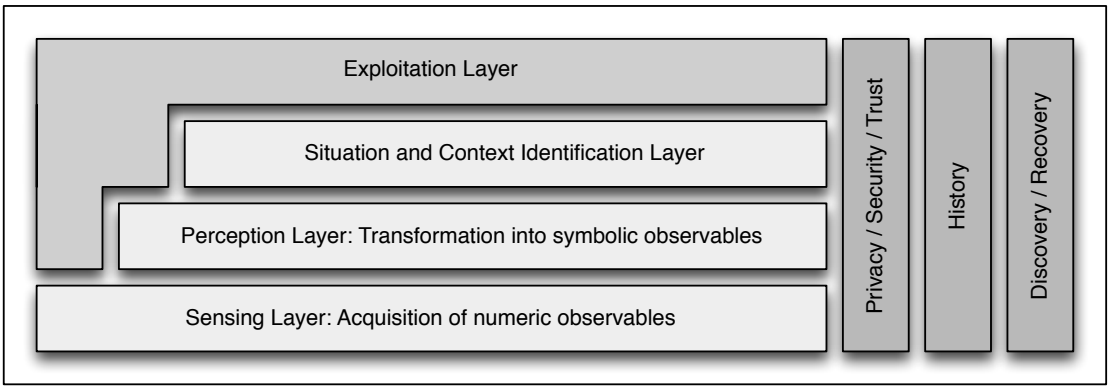


FIGURE 2.3: Reference architecture of context-aware computing systems for ubiquitous environments[13]

symbolic observables created in the preceding layer. Additionally, it tries to identify context transitions, i.e., alterations in contextual information that represent a movement from the current situation to a subsequent situation and anticipates such conditions to system or user-centric needs. The *exploitation layer* finally offers external applications access to the services and functions of a context framework. It acts as an adapter or middleware situated between the framework and applications that try to make use of the framework's functionalities. Requests and queries issued to the framework are processed in this layer.

The orthogonal layers represent aspects that must be incorporated (to a certain extend) in each of the horizontal layers. Hence, every vertical layer must include mechanisms that address trust, security, and privacy issues of context processing as well as context history, and context discovery and recovery. Context history, for instance, allows for tracing back context changes and infer on how a specific context representation has evolved over time. Combined with feedback mechanisms, historical context analysis further helps in elaborating on how well a context model adapts to user activities.

2.4.3.1 Context Acquisition Architectures

Context can be acquired from a multitude of different sources, e.g., by applying sensors or sensor networks, by deriving information from the underlying communication infrastructure or network, or by acquiring status information and user profiles directly from the device the user currently uses [BDR07]. In general, every piece of information or any content can be considered context-relevant if it provides information related to the user's current tasks and activities [WX06]. Therefore, context acquisition should not be restricted to capture context merely from hardware sensors. Instead, additional information sources should be integrated into the acquisition process. A social bookmarking service for example could be requested for obtaining information related to the user's interests or information needs. Especially when considering Web 2.0 applications in which users actively contribute, such services likely contain valuable information that can be used for complementing sensorial data [WX06].

In the following, we provide an overview of the types of potential context sources relevant for mobile devices, which range from hardware or software sensors deployed directly on the device, over devices located in the physical (ubiquitous) environment surrounding the user, towards external context sources such as Web services, Web APIs, and specifically Linked Data repositories.

- *Analysis of System and Execution Environment*

  A background service running on a mobile device can monitor system processes and associated threads to deduce information about the currently running applications and corresponding information needs of the user. Additionally, such service can track and analyze user queries to infer on the topics and information items a user is interested in. Tracing and aggregating that information enables the creation of information clouds similar to the concept of tag clouds in which more relevant and thus important items are highlighted. Explicit information according to the user's scheduled tasks, calendars, and preferences can be acquired via software sensors that encapsulate the corresponding interfaces to such data sources.

- *Internet and Online Communities*

  The Internet is a great source of context information [WX06]. With more and more users contributing to social communities such as *Facebook*, *Flickr*, *LinkedIn* (to name just a few) those portals provide a multitude of different information about a user. Due to the increased availability of such web applications that can be requested or accessed using mobile devices, the popularity and relevance subsequently becomes more apparent. Social bookmarking services such as *Delicious*[14] holds information about the preferred websites a user is interested in as well as the tags (terms) they use for annotating a particular website. With the advent of Web 2.0 applications, context sources available on the Internet are getting richer and increasingly important for context acquisition.

- *Local and Embedded Sensors*

  The continuing technical progress of current mobile devices together with the ongoing miniaturization of microprocessor-embedded devices[15] are reasons why more and more sensors such as accelerometers, digital compasses, etc. are deployed on mobile devices. Such sensors provide data about the physical environment in which a device operates. In newer operating systems such as the Android platform, such sensors are more tightly integrated into the operating system and offer dedicated interfaces and libraries for their utilization. However, some mobile system architectures and operating systems respectively only allow for utilizing a restricted set of sensors. This is due to the fact to not allow 3rd-party applications to have full access to the full range of deployed and available sensors so that premium services can only be offered by the manufacturer. In Java ME[16] (formerly J2ME) for example, applications are executed within sandboxes to which a number of restrictions are imposed. Such restrictive application models make it difficult to retrieve an exhaustive set of contextual data from embedded sensors.

- *Network Services*

  Some network providers offer additional services that may also serve as context sources. For example, a network provider may provide information about the current radio cell in which a mobile device was recognized. That information can be transformed to GPS-coordinates to derive information about the current location of the user. Another prominent example are Smart-ITs [GSB02, BG03] in which autonomous devices are equipped with sensor and sensing technologies as well as processing and communication capabilities that enables them to interact in an ad-hoc manner in order to exchange the context of artifacts. Such devices operate independent from a concrete technical or network infrastructure and can be considered as ancestors of ubiquitous devices.

[14] The online bookmarking service Delicious: `http://delicious.com/`

[15] However, cost is another factor that leads to equipping mobile devices with even more sensor devices but it will not be considered any further here.

[16] J2ME: `http://www.oracle.com/technetwork/java/javame/overview/index.html`

- *Sensor Networks and Ubiquitous Environments*

  Sensor networks are primarily used in ubiquitous environments for context information retrieval. Such sensors implement raw-sensing techniques to detect and measure aspects of the physical environment in the form of proprietary values such as a user's location, physical conditions such as temperature and humidity, acceleration etc. Those networks are denoted by [ATH07] as *rudimentary knowledge measurement engines* using sensor-based context detection. However, sensor-based context detection techniques may range from simple sensing technologies to more elaborated and sophisticated techniques by deploying full-purpose context sensors (cf. [BC04] who developed the *sentient object model*) or by combining several sensors to form a sensor network. This can be accomplished by combining several context information retrieved from selected sensors of a sensor network. Extensive work in this area has been carried out within the Smart-ITs-project (cf. [BG03, HGK+04]).

Some sensors, however, use proxies to provide their information. Physical sensors in general do not have sufficient computational resources and logics for providing high-level context representation and managing contexts. Therefore, the concept of dedicated *context proxies* are proposed that interpret, manage, and disseminate data provided by physical sensors and offer interfaces for accessing remote data sources that contain context-relevant data [EPR08]. This aspect should be considered as a first-class requirement for context processing architectures, especially due to the distributed and heterogeneous nature of ubiquitous environments and the sensors deployed in, where a context framework has to support such distributed and heterogeneous operational environments in most flexible ways [EPR08]. This results in the requirement of an ad-hoc and dynamic deployment of new context sensor proxies without the necessity of adapting context-aware applications to utilize newly integrated context sensors.

The way contextual data are acquired significantly influences the architectural style of a context-aware system [BDR07]. Context acquisition architectures can be broadly classified into three different groups (cf. [Che04]): (1) *proprietary architectures* that directly access locally deployed sensors where sensor and driver logic is directly implemented in application code limiting context reuse and exchange, (2) *middleware infrastructures* which employ a layered system architecture (e.g. [BKL+08a, CJ04, LFWK08]) encapsulating sensor specifics in dedicated components and expose uniform interfaces for context utilization, and (3) *context server architectures* (e.g. [HSP+03]) that operate similar to database management systems and offer remote access to contextual information hosted within a context repository. A detailed discussion about the advantages and limitations of each architectural style can be found in [CFJ03] and [Che04]. In the following, we provide a more detailed overview of each architectural approach:

- *Direct sensor access* is the primary option for gathering contextual data from hardware sensors embedded in a device. A context framework usually employs a dedicated layer that abstracts over concrete sensor specifics and encapsulate sensor-related driver logic. Layer-based approaches can primarily be found in newer approaches (e.g. [CJ04, BKL+08a, LFWK08]) since early context frameworks usually employed hard-wired sensor-logics directly in application code [Dey01, DAS01].

- *Middleware infrastructure* extends the idea of direct sensor access by employing a layered system architecture a priori where the methods for accessing context acquisition components are encapsulated in dedicated components and layers, respectively. The rationale is to conceal low-level sensor-logic from context utilization components, so-called context consumers or context services, and the framework as such to facilitate re-use and extensibility of context acquisition components. A middleware infrastructure is usually deployed in

distributed acquisition architectures to provide uniform interfaces that encapsulate specific sensor APIs and sensor details.

- *Context Server* represents a technical infrastructure that offers multiple clients remote access to contextual information hosted within a central context repository. Contextual information, which is stored in remote data sources can be exploited by clients in a distributed manner. A context server acts similar to a database management system since it handles concurrent requests and keeps track of modifications and context updates. The context server might also offer different representation formats of its contextual data depending on the protocol a client uses for requesting context information. Extensive processing tasks are handled by the context server itself rather than on the requesting components. Such an approach also allows for establishing load balancing algorithms in order to efficiently exploit the technical infrastructure.

Since context awareness in general is concerned with a multitude of different information sources and information types, its implementation in information systems requires an adequate middleware support (cf. [FMGI06]) to make such information available for context consumers in a controlled and well-defined way, where functions of the middleware should be by itself context-aware (e.g., context-dependent service discovery and interaction) (cf. [Sat02, CDA07]). A middleware-based architecture connects context acquisition components with other components of a framework or context consumers such as context services or external applications.

A middleware is defined as an abstraction layer situated between the operating environment of a software system and the applications operating in a distributed environment [Kja07]. The objective of a middleware—especially for traditional distributed systems—is to unify access and exploitation of distributed data sources by encapsulating the heterogeneity and distributed architecture of the underlying system infrastructure. This has proven to be useful for wired and static environments but fails to compete in dynamic and wireless environments due to the decision making processes of the running applications that are often based on technical and environmental facts [Kja07]. Middleware architectures for context-aware systems are used to deal with sensor heterogeneity and context source diversity. Their objective does *not* lie in hiding sensors specifics but to provide suitable abstractions and interfaces for their exploitation, and to offer technical and architectural aspects of the underlying system or network infrastructure as (*environmental*) context information. Context-aware middleware systems range from pure middleware systems to complete computational infrastructures capable of managing the surround physical environment and can be classified according to their functional range and the capabilities they offer[17].

2.4.3.2 Context Management Architectures

Two directions have been identified for managing context information [FMGI06]: the first direction is denoted as *pervasive spaces* and direct the management of contextual information to the physical environment where such information is gathered from. In such environments, a centralized system is deployed for context management (cf. context server architecture) that can be accessed by context consumers. The second direction is based on the concept of *delegation* where context information management is placed on end-user devices, which resembles a decentralized architectural approach. Hence, a device is responsible for the entire context processing and management life-cycle, although devices can make use of distributed context

[17] See [Kja07] for a classification taxonomy distinguishing between the type of technical infrastructure, storage facility, reflection mechanisms, quality of context data, composition of context services, migration types, and adaptation support

repositories or ubiquitous sensors for context acquisition. Such devices acquire context autonomously and operate independently from each other. They collaborate in an ad-hoc manner and connect to its peers for context sharing and exchange. They use technologies for sharing contextual information among each other in a distributed and autonomous fashion (e.g. [MC02, GSS02, BC04, HIMB05, RF05, PvHS07, RSP07, HDW09]), which introduces the necessity for *context consistency control*, *context replication*, and *context dissemination* strategies (e.g. [CDY90, Len96, RSP07]).

However, different classification schemes for context management architectures have been proposed by, e.g., [DAS01, BDR07, EPR08]. These works distinguish between a (1) direct integration of context management functionality into context-aware applications, (2) context management services, and (3) context-aware devices and services augmented with context management functionalities. In the following, we give a brief overview of each architectural approach:

- *Direct Integration into Context-aware Applications.* This group of frameworks is characterized by a direct communication between context acquisition components such as sensors and context-aware applications, the so-called context consumers (e.g. [DAS01]). Applications need to know in advance which sensors are available and how to communicate with them. As a consequence, context data semantics are limited to the applications in which they were interpreted and can, in most cases, not be shared between applications which makes it difficult to integrate new sensors dynamically in running frameworks. Sensors have to be queried by each application separately, which impede the process of context aggregation, exchange, and dissemination.

- *Using Context Management Services.* Within that approach, context information is gathered in a single place (e.g. a central component within a system or the environment) and forwarded to the respective context-aware applications or context consumers either on request or proactively. The advantage of this approach lies in the central aggregation of contexts. At the same time, this is one of the major drawbacks of this approach since it is contradictory to the nature of context per-se [EPR08].

- *Augmenting Devices or Services with Context Management Functionality.* The third class of frameworks is grounded on the idea of embedding context-management functionalities directly into devices or context services whose functionality can be shared across components (e.g. [BC04, EPR08]). Each sensor or context service subscribes to a registry in which it is registered together with the context capabilities it offers. Whenever there is need for a particular context service, a context consumer requests the service registry and gets the corresponding service forwarded. Thus, new services can be easily integrated and utilized. The context management framework proposed in this thesis adheres to this approach since every context acquisition component contains individual functions for processing and interpreting context-relevant data.

Context-aware functions, in general, can be classified according to their responsibilities for (i) representing information and services that enable the adaptation of user interfaces and provision of application services with respect to contextual constellations; (ii) triggering specific application or device behaviors automatically; for instance, a car navigation system is going to compute a new route in case some significant contextual parameters such as congestion alerts or construction works have been provided; (iii) adopting data retrieval algorithms; for example, queries to remote data sources can be adapted according current information preferences of a mobile user [ADB+99]. The context framework presented in this thesis can be ascribed to the last category.

Korpipää [KMS+05] claims for a *direct management* of rapidly changing context data on the device itself, leading to independent, autonomous context acquisition and processing. By directly integrating context management functionality into mobile applications, such applications need to know in advance which sensors are available and how to communicate with them. As a consequence, context data semantics are limited to the applications in which they were interpreted and can, in most cases, not be shared between applications which renders a dynamic integration of new sensors in running frameworks difficult. Sensors have to be queried by each application separately, which impede the process of context aggregation, exchange, and dissemination. In contrast, if context management functionality is implemented in devices or sensors, their functionality can be shared among components and integrated during run-time (e.g. [BC04, EPR08]). Other works (e.g., [KHK+04, KMS+05, RSP07]) apply the concept of *End User Development (EUD)* [BCR04] to the development process of context-aware functions and features, which in general should be supported by easily amenable tools in order to encourage users to customize application behavior and implement individual context-aware features.

2.4.3.3 Context Processing and Utilization Architectures

However, most of the proposed context-aware computing infrastructures (e.g. [DAS01, GSB02, HIMB05, EPR08]) are built on the idea of separating the process of context retrieval (context acquisition) and context processing from the underlying application logic through the provision of suitable abstraction mechanisms; this aspect is denoted as *separation of concerns* [DAS01]. Literature research reveals that there exists three different context management models for the coordination of context processes and context consumers [Win01].

- *Widgets* as proposed by [Dey01, DAS01] are software components that encapsulate or abstract over specific driver or sensor-logics and offer public interfaces for utilizing a context sensor. Widgets resemble the so-called concept of *separation of concerns* (cf. [DAS01]) and are intended to increase the reusability among context services by encapsulating low-level sensor specifics. They are controlled by a central authority, the so-called *Widget Manager*. Additionally due to their self-contained and autonomous nature, widgets can be easily replaced by components that provide the identical kind of data but offer a higher Quality of Service (QoS). The tight coupling of widgets with a context management's infrastructure increases acquisition efficiency but also entails a loss of robustness in case of malfunctions [BDR07].
- *Networked services* are based on a client-server architecture and resemble the Context Server approach[18]. They represent a more flexible approach in processing context information and are usually used for connecting higher-level components. This approach became prominent with the advent of network-based services and distributed system infrastructures. In contrast to the Widget-based model, networked services primarily use distributed discovery techniques for accessing network services and expose a more complex structure compared to Widgets since they contain functionalities for establishing network connections, message handling, and error handling. Networked Services are not as efficient as Widgets in terms of context acquisition but provide greater robustness with respect to failures [BDR07].
- The *Blackboard model* represents an asymmetric approach for data sharing and resembles a data-centric perspective. The central component of this approach is the so-called *blackboard* that represents a message board to which context components can send messages

[18] cf. Section Architectures

in an asynchronous way. Context consumers subscribe to a context blackboard in order to receive notifications whenever a new message has been sent to the board. Context sources can be easily added, which is one advantage of this approach. Subscription to messages follows a pattern matching approach, which varies among different blackboard implementations and may, for instance, be based on "sophisticated inference algorithms" tuple-based representations, or a "field-by-field comparison" of tuples [Win01]. Although context blackboards allow for a simple integration of additional context sources, a central server for hosting the blackboard and handling communication concerns is still required accompanied by a loss of communication efficiency [BDR07].

In summary, Widgets provide a process-centric perspective, Networked Services a service-centric perspective, and Blackboards a data-centric perspective [Win01]. Networked services provide a greater flexibility compared to the widget-based approach as it uses several discovery techniques for finding network services and resembles the context-server approach, although with an accompanying loss of efficiency [BDR07].

2.4.4 Key elements of Context Processing and Management Frameworks

Several context management and processing frameworks for mobile and pervasive computing information systems have been proposed that cover some of the following aspects: context detection and acquisition, context identification and representation, context retrieval, context source discovery and integration, context augmentation, aggregation, and reasoning, as well as context storage, dissemination, and service adaptation (cf. [ADB+99, CK00, SP04, RTA05, BCQ+07, BDR07, Kja07, SB08a]). Those frameworks incorporate a number of functional entities that can be classified along the general functions of generation, management, and application[19] of contextual information [FMGI06]. A number of works defined requirements a context framework or context-aware application must fulfill to be classified as context-aware. These requirements serve as a basis for defining a number of *key elements* and *functionalities* a context framework must exhibit. In the following, we give an overview of the most relevant features:

- *Context acquisition* refers to the set of methodologies and techniques of acquiring context-relevant data from context sources, which might be a locally deployed sensors, ubiquitous devices, or Web-based services and applications. Context acquisition can be divided into four sub-phases (cf. [KMK+03]): (i) *sensor measurement* for retrieving raw sensorial data, (ii) *preprocessing* in which data arrays of measured sensorial data are created for chronological quantization and feature calculations, (iii) *feature extraction* to calculate more specific context features such as classifying ambient noise values according to predefined classification features using specific mathematical and logic-based methods[20], and (iv) *quantization and semantic labeling* for aligning context features to real-world contexts where their specific meaning is represented by concepts and properties from controlled vocabularies and ontologies. In some other works, this process is also denoted as *context recognition*, *context detection*, or *context gathering* [KMK+03].

- *Context aggregation* denotes the task of combining contextual information from different, potentially distributed and heterogenous sources in order to derive higher-level context

[19] That is responses according to contextual information processing.
[20] See Figure 2.4 for an illustration of quantizing extracted features from acquired contextual information.

information that was not anticipated at design time of a system. To facilitate the aggregation of contextual information, acquired context descriptions must adhere to a flexible and well-defined context model that forms the basis for its composition. However, a problem of context aggregation, in particular of low-level contexts, is that of *competing context sources* [LFWK08], which might introduce inconsistencies that arise from aggregating context information that has been acquired from similar sensors, i.e., sensors that measure identical real-world aspects (e.g., in-house versus outdoor location ascertainment using positioning devices with varying precision and resolution). A strategy to handle such conflicts is to explicitly embed users in evaluation processes in order to manually specify the trustworthiness of context sources where contextual data coming from a source with a higher ratio can overrule those from lower rated sources [LFWK08]. Another strategy is to assign confidence parameters to context sources, but this requires a-priori knowledge regarding a potential context source, which is contradictory to the idea of ubiquitous computing environments [EPR08]. In other works (e.g. [GSB02, BC04]) context aggregation is also denoted as *context clustering* or *context fusion*.

- *Context consistency* is an advanced form of context aggregation that specifically focus on the aggregation of distributed context models that are subject of frequent changes. Context consistency ensures that the aggregated context models adhere to the structure and semantics of the global context model or the prescribing schema. Representing contextual information using Semantic Web languages facilitate the tasks of detecting and resolving inconsistent contextual information that result from *imperfect context sensing*, that is, indirectly acquired contextual data might be inconsistent, incomplete, or outdated [ATH07].

- *Context discovery* describes the process of locating, integrating, and exploiting context sources, which are usually dispersed across the environment and not known at design time. It must take into consideration the service descriptions and protocols exposed by a context source. This feature is particularly important for context frameworks in the domain of ubiquitous computing due to the difficulties in discovering and integrating ubiquitous sensors resulting from the technical diversity of devices as well as the interfaces, protocols, and data formats such devices expose. Especially at design time, such information is usually not available and it can not be guaranteed that a context source is compatible to a context management framework. Context discovery as such is not always a straight-forward process since both low-level and high-level contexts "incorporate some uncertainty in form of inferred probability based on previously learned evidence" and "often ambiguously reflect real-world conditions" [KMK$^+$03]. Other approaches (e.g., [BS03, BTC06, WX06, PvHS07, SWvS07, MTD08]) therefore suggest to align *Quality of Service (QoS)* or *Quality of Context (QoC)* parameters to context sensors and contextual information to support the process of selecting appropriate context sources.

- *Context query and dissemination* is used for requesting information from context sources such as sensor proxies or context data repositories. Therefore, the different heterogeneous context descriptions must adhere to a general and global context model so that queries can be issued against it – ideally in a transparent manner. Context data can be requested either by *querying* them or by using a *notification mechanism* to inform interested components about the availability of updated data (consumer-subscriber model).

- *Context adaptation* describes the system's capabilities to adapt its behavior according to contextual constellations. System configurations should automatically adopt to respond intelligently. For instance, evaluation of context changes is often realized by using conditional rules (cf. [BC04, KMS$^+$05]. Adaptability requires that the user context is *self-descriptive*,

that is, user context is described in terms of constraints, preferences, believes of the user's needs so that services can be located and executed on behalf of where a context framework acts as a *mediator* between services, user needs, and the environment [ATH07].

- *Context reasoning* allows for deriving new context-relevant knowledge that is not explicitly asserted or anticipated beforehand within an instance of a context description or context model. Additionally, reasoning can be used to perform consistency checks to elaborate whether a given contextual statement holds true or not. Reasoning is also used for deriving higher-level contexts.
- *Context quality indicators* allow for computing the trustworthiness and validity of a context description. This is of significant importance for acquiring contexts from distributed and heterogeneous context sources, as it is the case in mobile ad-hoc or ubiquitous scenarios. Such distributed and heterogeneous context sources usually lack a global model or scheme for describing the data they offer. Therefore, a considerable amount of works (e.g. [BS03, BTC06, PvHS07, SWvS07, MTD08]) has considered *Quality of Context* (QoC) to handle the diversity and heterogenous nature of context sources in ubiquitous and mobile environments.

For calculating context features from low-level context data and binding it to real-world context, two quantization methods have been proposed by [KMK+03]: (i) the definition of *crisp limits* and (ii) the usage of *fuzzy sets*. An example of these methods is depicted in Figure 2.4 where the x-axis represents the value of a context data element (the measured value x) and the y-axis the membership function $\mu(x)$[21].

Let X^S denote the set of context data values that correspond to a sensor S; let x denote a concrete context data value retrieved from the sensor S where $x \in X^S$. Let F^X denote a feature classification for the context data set X that consists of a number of classification intervals or labels f_i with $F^X := [f_1, f_2, \ldots, f_n]$. Hence, by applying the fuzzy membership function $\mu(x)$, a context data value x is mapped to the interval $[0 \ldots 1]$ for a given context feature classification $f_i \in F^X$:

$$\mu(x)_{\text{Feature } f_i} \longmapsto [0 \ldots 1] \tag{2.1}$$

Crisp limits evaluate context features according to `true` or `false`-values and allow for a distinct differentiation between context value classifications calculated by the fuzzy membership function $\mu(x)$ that returns values of either `0` or `1` indicating that a given classification feature f_i is true for a sensor value $x \in X^S$ or not (cf. Figure 2.4 (a)). For instance, the humidity classification features 'dry', 'intermediate', 'humid' represent high-level context classifications to which the value of a given context sensor is mapped if the measured value lies within a predefined range and is true for the corresponding membership function $\mu(x)$. The conforming high-level context concept (context feature) is then included in a context description for the given feature (e.g., 'humid' in case humidity value is $>$ `0.80`). Hence, only one membership function $\mu(x)_{f_i}$ and classification feature $f_i \in F^X$ respectively can be true at one point in time.

$$F^{\text{Humidity}}(x) \oplus \begin{cases} f_{\text{dry}} &= 0 \\ f_{\text{intermediate}} &= 0 \\ f_{\text{humid}} &= 1 \end{cases} \tag{2.2}$$

[21] This section represents an extended version of the example proposed in [KMK+03], page 43ff.

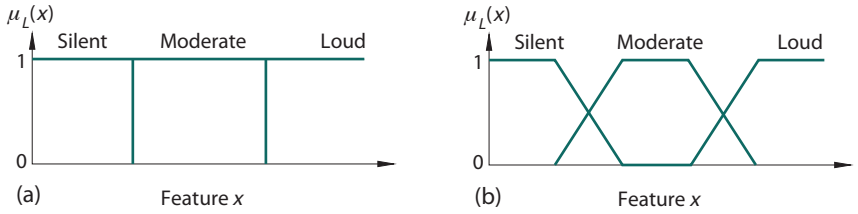


FIGURE 2.4: Crisp (a) and fuzzy (b) quantization methods for calculating context features from low-level context data[22]

Fuzzy sets, in contrast, allow for specifying transitions between feature classifications $f_i \in F^X$ and enable a continuous quantization of context values as illustrated in Figure 2.4 (b). For instance, the membership function of a fuzzy quantization F^{Sound} of noise levels consisting of '`silent`', '`moderate`', '`loud`' could return $\mu(x) = (0.7/f_{\text{silent}}) + (0.3/f_{\text{moderate}}) + (0/f_{\text{loud}})$ for a given decibel value x.

$$F^{\text{Sound}}(x) \oplus \left\{ \begin{array}{rcl} f_{\text{silent}} & = & 0.7 \\ f_{\text{moderate}} & = & 0.3 \\ f_{\text{loud}} & = & 0.0 \end{array} \right. \tag{2.3}$$

Figure 2.5 illustrates an example for the deduction of higher-level context information from context feature classifications. In this example, naive Bayesian classification [FL04] is used for the aggregation of low-level contextual data to determine whether the user's current position refers to an indoor or outdoor location. The white rectangular boxes represent context features F^X that correspond to elements of a context ontology (the context types), which have specific context feature classifications $f_i \in F^X$ assigned (the beige boxes). The confidence values returned by the membership functions $\mu(x)_{f_i}$ of all context types (context feature classifications f_i) are displayed in the brown boxes. In this examples, the context feature classification values are mapped to the upper-ontology classes '`Indoors`' and '`Outdoors`' based on the confidence values calculated by the corresponding membership functions[23]. Hence, an aspect of a given situation (in this case the position of the user and her device) can be classified according to the two upper-class context ontology types with respect to the previously calculated confidence values. Bayesian classification is used for the aggregation of the context feature classification instances and corresponding confidence values into pre-defined high-level context ontology classes. Naive Bayesian classifications in general provides rather satisfying results in recognizing specific situations under controlled conditions, but can only be applied in restricted and pre-defined settings [KMK+03].

For the representation of both low-level and high-level contextual information, a context framework must support *multi-granular* context descriptions that incorporate different levels of complexity according to the contextual information contained in a description [ATH07]. Those descriptions consist of low-level sensorial data as well as high-level contextual information referring to a user's current situation represented through concepts from higher-level ontologies; for instance, a user's location might be represented as *'at work'*, *'at home'*, or a specific room number in a building. The spectrum of context-relevant information ranges from simple binary arrays to detect whether a sensor or device is turned on or off, to sensors that elaborate on the activities a user is currently involved [KQJH03].

[22] Taken and adapted from [KMK+03], page 46.
[23] Please note that both types of membership functions are used: fuzzy-logic as well as crisp transitions.
[24] Taken and adapted from [KMK+03], page 47.

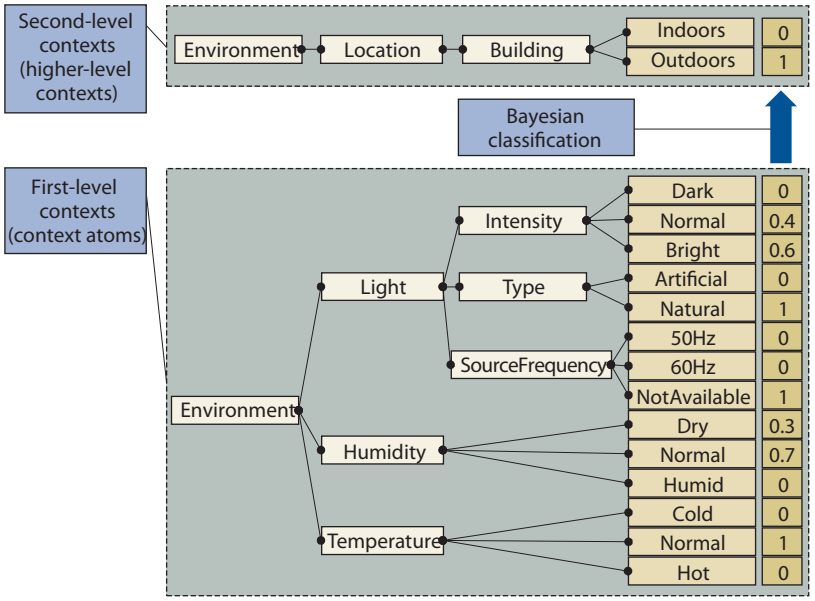


FIGURE 2.5: Example of low-level to high-level context-aggregation using Bayesian classifications[24]

In general, a context management framework should include high-level context information, so-called *context abstractions*, in its context descriptions since there exists a direct relationship between the level of abstraction and the motivation of adoption and usage [DAS01]. As a consequence, context should be shifted towards the notion of situation as a higher-level abstraction of context since context is always defined relative to the situation in which it is acquired (see, e.g., [LFWK08, Geh08, SWB+08, THS09, CCMS10] for related approaches). A context framework must therefore allow for the specification of complex situations referring to real-world events that can not be derived from single context providers or sensors. A number of works (e.g. [LMWK05, LFWK08]) therefore applied situational reasoning as a more sophisticated form of context awareness for the aggregation of several context information fragments that were harvested from multiple context sources. In situational reasoning, the explicit knowledge encoded in context ontologies is used to infer additional high-level qualitative context assertions based on the recognition of specific contextual constellations that represent a given situation. Hence, a context description is complemented and enriched with additional inferred context statements on a qualitative level.

In the following, we illustrate how situational reasoning based on classification rules can be performed by means of a concrete example proposed by [LFWK08]. Figure 2.6 depicts a fragment of a situational ontology with a top-level concept denoted as '*Situation*' that is further refined by the concepts '*Private*', '*Business*', and '*Meeting*' which refer to elements (concepts and relations) defined in a situational ontology [LFWK08]. By applying dynamic assertional classification of situation descriptions, the concepts defined in context ontologies can be mapped to an upper-level situation ontology where a situation can be classified as "Business Meeting" when the individuals of a context ontology match the corresponding rule of a business meeting.

In a concrete example, a situation of an individual is classified as business meeting ('*Business_meeting*') when the conditions for the concepts '*Situation*' and '*Meeting*' hold true and

[25]The situation ontology was originally presented by [LFWK08], page 14.

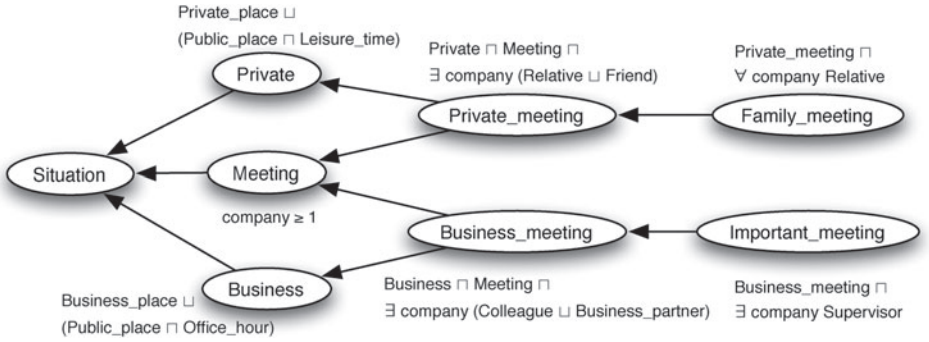


FIGURE 2.6: Example of a situation ontology that classifies situations as business and private[25]

there a people classified as colleagues ('*Colleague*') or business partners ('*Business_partner*') in an individual's immediate environment:

$$\text{Business\_meeting} := \text{Business} \sqcap \text{Meeting} \sqcap \exists \text{company}(\text{Colleague} \sqcup \text{Business\_partner})$$

Additionally, the following example represents a classification rule that classifies a situation as a business situation, if it takes place at a location that is either classified as a business place (e.g. an office or a conference room) or a public place (e.g. an airport or a railway station) in case it is taking place at office hours:

$$\begin{aligned}\text{Business} := \text{Situation} \sqcap (&\exists \text{location.Business\_place} \sqcup \\ &(\exists \text{location.Public\_place} \sqcap \exists \text{time.Office\_hour}))\end{aligned}$$

However, in the approach proposed by [LFWK08], the classification of situations is restricted to high-level qualitative context elements. Low-level contextual data are represented using an XML-based meta model and managed separately due to size of the produced context models, which may become significantly large and decelerate reasoning and processing performance. The use of ontologies for representing contextual information on mobile devices is problematic since available reasoners are generally weak in handling large RDF documents or models efficiently [LFWK08]. Mobile devices in general provide less computational power and memory capacity than desktop computers or servers wherefore we recommend to keep context descriptions relatively small with an average amount of triples that do not exceed a maximum of 2.000 or 3.000 triples (see Chapter 6). However, reasoning performance has been significantly increased within the last years through the use of multiple optimization techniques for handling large amounts of data [LFWK08].

2.4.5 Summary and Discussion

The usage and utilization of the notion of context in information systems raises some technological as well as human-related challenges. Dey et al. [DAS01] identified the poor understanding on what context constitutes, how it is to be represented in information systems, and the lack of conceptual models, methods, and tools that would promote the design of context-aware mobile applications as one of the main reasons why context awareness has only insufficiently found its way into the essence of mobile and ubiquitous computing yet. This additionally hampers empirical investigations in human-computer interaction and interaction design. Effective interaction

design thus requires the multiple dimensions of context to be reflected in mobile application design and implementation [RTA05].

The fact that context is often misinterpreted, superficially used, and lacking an unambiguous and widely agreed definition are some of the reasons, why context awareness has not found its way into the corpus of most mobile applications yet [RTA05].

Many works indicated the non-existent availability of a general model of context and context awareness as one of the main problems of context-sensitive systems. This fact in particular concerns mobile computing where the notions of context and context awareness are used ambiguously across communities and reflect specific application domain peculiarities (cf. [BCQ+07, RTA05]). This problem also hampers context-aware application development for mobile systems since a widely-accepted and well-defined programming model does not exist wherefore sensor-logics are often hard-wired into application code and application developers have to deal with low-level interactions between sensors and context-acquisition components [DAS01, BC04]. Therefore, the semantics of contextual information are limited to the applications in which they were acquired and are represented using proprietary formats for representing contextual information. This exacerbates the exchange and dissemination of contextual information and binds its utilization to the application they stem from. In general, context information should be sensed and gathered from a variety of different sources that are usually distributed across system boundaries. This requires open and well-known standards for describing contextual information as well as interfaces and protocols for distributing that information[26]. Newer approaches (e.g., [BC04, MFC07, BKL+08a]) employ more flexible designs for context processing and representation where sensor-logics or sensor-specific APIs are encapsulated in specific components that can be mutually shared or employ middleware infrastructures (e.g., [HIMB05, HM05]) for facilitating communication and interoperability between context processing components while using knowledge representation languages from the Semantic Web such as RDFS or OWL for representing contextual information [PvHS07, MT07].

2.5 Problems of Context-aware Computing

A fundamental problem of context-aware computing is that of *context ambiguity* [Dey01] and *context imperfection* [HIR02] which refers to the implicit assumption shared by many context-aware computing approaches that the computational context is a 1-to-1 reflection of the real-world context. Evidently, this assumption is wrong since the way context is conceived by individuals differs substantially from the way it is acquired and represented electronically [Dey01, Dou04]. A logical consequence of that misperception is that a context framework can only work on a more or less accurate context representation where the degree of accuracy is determined by numerous technical and soft factors. The unpredictable and relative nature of context renders a determination of all contextual aspects that constitute a specific context at a system's design time difficult, if not impossible, since context is always defined relative to the situation in which it is used. An electronic representation of context can therefore never be universal in that a context model contains all information that characterize a given situation; instead it only represents a relevant subset of the constituting real-world context [Dey01, HIR02, Dou04]. The problem is that a 1-to-1 relation between a situation and the describing context information does, in most cases, not exist wherefore a situation can be represented by multiple context models with a specific degree of accuracy w.r.t. the several viewpoints. Several methodologies such as bayesian networks,

[26] In Section 2.8, we demonstrate the different possibilities the Semantic Web offers for modeling and distributing contextual information in mobile ad-hoc and highly dynamic environments.

case-based reasoning, stochastic models, or machine learning techniques have been proposed for defining precise transitions between different context descriptions with as little ambiguity and overlapping as possible. However, such approaches contribute towards increasing the accuracy of context acquisition and context representation but do not help in identifying all constituting aspects of a specific situation. Context-aware computing in general is merely an approximation to a real-world situation rather than a 1-to-1 reflection of it.

Three complementary approaches have been proposed by [DAS01] to deal with context ambiguities:

1. *Let applications choose an appropriate way on how to deal with ambiguities*

   Within the first approach, an application can specify the accuracy of the context information it requires. On the other hand, this requires the context framework and the involved applications to use the same accuracy measurement method or at least compute and share measurements on a common basis. In decentralized context processing architectures where context sensing components are distributed, this requirement becomes more complicated to fulfill since a framework has to guess on the accuracy of acquired contextual data. Consequently, a context sensor has to be annotated with information that allows a context framework to ascertain the accurateness of its context data, which on the other hand also depends on the context consumers themselves that make use of the acquired context information. For one application, the acquired context might be sufficiently accurate whereas another application requires a more precise representation.

2. *Automatic disambiguation of context*

   Another common approach for resolving or disambiguating contextual data is to collect context data from as many sources as possible to improve its accuracy. Collecting contextual data from multiple, distributed sensors is per-se a non-trivial task since it requires a certain degree of compatibility in terms of the protocols used for communicating contextual data as well as on syntactical, structural, and semantic level, i.e., how to interpret a certain contextual value or property. The process of combining multiple (often homogeneous) context sensors is called *sensor fusion* (cf. [GSB02, BC04]). Although sensor fusion is a valid strategy for reducing context ambiguities by increasing its accurateness, inherently given ambiguities can not be fully eliminated by automated approaches.

3. *Manual disambiguation of context*

   A third approach is that of manual disambiguation of context data where the user is involved in the process of interpreting contextual data and resolving disambiguations. In this approach, the system should support the user by providing alternatives and give feedback on the consequences of their chosen option. Ideally, this information is transferred to or stored by the context framework so that other components can make use of it or use it for future actions.

Another problem of context-aware computing is that most architectures are targeted towards a specific application or domain [EPR08]. The difficulties hereby are that certain high-level context interpretations are not absolute characterizations per se, since the concept 'high temperature' for instance depends on the context or situation in which it was acquired. This dependency makes it difficult to share context information in an application and domain-independent manner since implementing new application behaviors based on context characterizations made for one application might not be appropriate for another one [EPR08]. Some works [KA04, CX06, Teo08]

therefore focus on describing and representing context in an application independent manner by means of concepts from activity theory [Nar95, Kuu95] using collaborative plans [GK96], task analysis [RW03, Dia90], aspect-oriented context modeling and modularization [CCSC07], or situational reasoning [BKL+08a, LFWK08] to decouple context from specific application domains and provide abstract contextual concepts (e.g., "business meeting") that adhere to upper-level ontologies. Context and contextual information needs a uniform representation to be effectively managed, integrated, and processed by reducing the ambiguities inherently attached [FMGI06].

Early approaches suffered from the tight integration of context acquisition and context-aware features which are treated by applications in proprietary ways leading to inflexibility in application design and reduced extensibility with respect to the development of new context-aware functionalities [KMS+05]. The proposed context management and processing framework provides the necessary technical infrastructure on which additional more sophisticated layers (e.g., for situation-awareness) can be deployed. Such layers allow for aggregating the contextual artifacts acquired by the underlying framework and apply different methodologies (e.g., Bayesian networks, case-based reasoning, stochastic methods, etc.) for context interpretation, consolidation, and augmentation.

2.6 The Semantic Web

The *Semantic Web* [BLHL01] is the idea of expressing rich, machine-processable knowledge using the Web infrastructure. Descriptions about *resources* (which are entities of any kind, including digital objects like documents and media, physical objects like humans, cars, or buildings, and abstract concepts like locations, topics, and time periods) are published in a structured format and using vocabularies that follow a well-defined semantics. Applications can consume these descriptions, interpret them, merge them with descriptions from other sources, and infer new knowledge or determine the truth value of statements. In analogy to the World Wide Web, which nowadays serves as one underlying knowledge-provisioning infrastructure for a wide variety of human knowledge workers, the Semantic Web is envisioned to serve as an underlying information-provisioning infrastructure for information-centric applications, whereas the information processing is performed semiautomatically by computer programs.

Broadly, the Semantic Web consists of a stack of technologies that build on each other. The core technologies are shared with the World Wide Web: *Uniform Resource Identifiers* (URIs) [BLFM05] to identify resources, and *Hypertext Transfer Protocol* (HTTP) [FGM+99] for the transportation of information. URIs are a fundamental part of the Web architecture and are considered as a single global identification system for resources [JW04]. They allow for a globally unique identification of resources across the Web thus promoting *large-scale network effects* and are defined in [JW04] as follows:

> **Definition 2.1** (URI)**.** *A Uniform Resource Identifier (URI) is a compact sequence of characters that identifies an abstract or physical resource. [...] It enables uniform identification of resources via a separately defined extensible set of naming schemes [...]. How that identification is accomplished, assigned, or enabled is delegated to each scheme specification.*

On top of these core technologies, the *Resource Description Framework* (RDF) [KC04] is used as the abstract data model in which all information is represented. RDF is a triple-based graph

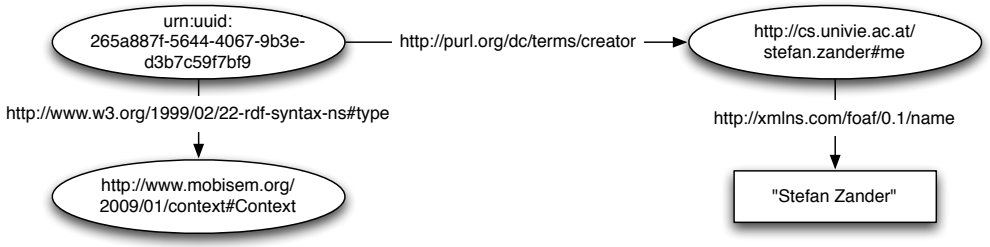


FIGURE 2.7: Example RDF Graph

model, with the *statement* being the atomic unit of information. Each statement consists of three elements (*subject*, *predicate*, and *object*), where the *predicate* identifies the relationship that is asserted to exist between the *subject* and the *object*.

Definition 2.2 (RDF Statement)**.** *An RDF statement $\mathcal{S}$ is a triple $\mathcal{T} = \langle S, P, O \rangle$ consisting of a subject S which is either a URI reference or a blank node, a predicate P which is a URI reference, and an object O that can either be a URI reference, a blank node, or a (plain or typed) literal.*

An RDF document consists of a set of RDF statements and can be considered as a *labeled multi-graph* [EPR08]. URIs can be used for all three elements; whenever the same URI is used in multiple statements these statements are referring to the same resource (or real-world entity). Therefore, a set of RDF statements that shares common URIs can be interpreted as connected graph (cf. Figure 2.7). Through the use of URIs as identifiers for information entities, these can be referenced from outside the system context and incorporated into external knowledge structures. RDF itself is an abstract data model; in turn, there exist several serialization formats that can be used to exchange RDF statements between parties, including *RDF/XML* [Bec04], *Turtle* [BBL08], *N3* [BLC08], *N-Triples* [GB04], and *RDF/JSON*[27].

Based on RDF a set of technologies has been developed that aims to make more "knowledge" out of the data represented in RDF graphs. Higher-level languages like *RDF Schema* (RDFS) [BG04], *Web Ontology Language* (OWL) [MPSP09], and *Rule Interchange Format* (RIF) [BHK+10] can be used to define the constraints of a domain of discourse based on formal logics; using these languages, valid combinations of statements can be defined axiomatically. This allows implicit knowledge to be discovered based on asserted information using reasoners such as *FaCT++* [TH06], *Pellet* [SPG+07], or *Racer* [HM01], to detect inconsistencies in knowledge bases, or to determine the truth value of statements, given a set of background knowledge. A query language (*SPARQL*) [PS08b], which resembles similarity to the SQL language for relational data, can be used to formulate structured information needs, which are evaluated against a set of RDF graphs.

One line of development within the ongoing Semantic Web research field is *Linked Data* [Biz09], which denotes the practice of publishing data on the Web according to simple core principles [BL06a]. Its core idea is to denote resources (which includes real-world entities as described above) exclusively using HTTP URIs, and to allow data consumers to directly de-reference these URIs (i.e., to fetch their representations using HTTP `GET` methods). Upon this de-referencing, structured information about the resources is returned, which contains links to other relevant

[27] RDF/JSON syntax specification: `http://docs.api.talis.com/platform-api/output-types/rdf-json`

resources. These other resources can then, in turn, be retrieved by the client, allowing it to navigate through a global information network based on the "follow-your-nose" principle.

Since 2007, when the Linked Data W3C community project[28] was established, a significant amount of data has been published according to the Linked Data principles. This includes popular data sets of general interest like *DBpedia* (consisting of structured information extracted from Wikipedia pages), geographic information (like *Geonames*), media-related content like *BBC Programmes*, and bibliographic information (e.g., *DBLP*). An example of a highly distributed data set is the entirety of all *FOAF Profiles*, which are usually served on private infrastructure, and are interconnected based on social relationships between their owners. Through the (partly indirect) interconnection of these data sets, light-weight data integration can be performed, and information about the same entities can be gathered and combined from heterogeneous, distributed sources.

2.7 Representing Contextual Information using the Resource Description Framework

In this section, we extensively discuss the implications of using RDF as a description framework for representing contextual information. An RDF document can be considered a set of distinguishable RDF triples, which describe resources as well as the relationships that exists between them using vocabularies that are defined on the basis of ontologies and whose elements can be identified using URIs. These elements form a *directed labeled graph* of (semi-)structured information, which is a very common and natural way to express domain knowledge [BLHL01, KC04]. RDF uses a graph-based data model for representing information that is independent of a specific serialization mechanism or format. It has been developed as a framework and language for representing meta data about resources in the Web [MM04]. In the context of the World Wide Web, a resource denotes or represents an item of interest that is identified by a Uniform Resource Identifier (URI) [Lew07]. In the W3C Recommendation "Architecture of the World Wide Web, Volume One" [JW04] a resource is defined as follows:

> **Definition 2.3** (Resource)**.** *The term "resource" is used in a general sense for whatever might be identified by a URI. It is conventional on the hypertext Web to describe Web pages, images, product catalogs, etc. as "resources". The distinguishing characteristic of these resources is that all of their essential characteristics can be conveyed in a message. We identify this set as "information resources".*

RDF uses the concept of Uniform Resource Identifiers (URIs) [BLFM05] for identifying both *information resources* as well as *non-information resources*, which are identifiable but not directly retrievable (cf. [JW04, BCH07]). In contrast to the concept of URLs [BLMM94], which not only identify Web resources but also provide information about locating and retrieving a resource's representation using specific protocols and network locations, URIs resemble a more general concept of identifying things on the Web that do not necessarily need to have a network location nor require a specific form of access mechanism (cf. [MD02]). URIs may also be used to denote abstract or intangible concepts such as "legislation period" or "law". In [Lew07] an information resources is defined as:

[28] W3C Linked Open Data Community Project: `http://esw.w3.org/SweoIG/TaskForces/CommunityProjects/LinkingOpenData`

> **Definition 2.4** (Information Resource)**.** *Information resources are resources, identified by URIs and whose essential characteristics can be conveyed in a message. The pages and documents familiar to users of the Web are information resources. Information resources typically have one or more representations that can be accessed using HTTP. It is these representations of the resource that flow in messages. The act of retrieving a representation of a resource identified by a URI is known as dereferencing that URI.*

Additionally, information resources usually have one or multiple representations that constitute the messages that are exchanged between applications or users [Lew07]. Information resources can be accessed using the Hypertext Transport Protocol (HTTP) [FGM+99] where *dereferencing* a URI denotes the process of retrieving a resource's representation. In contrast, non-information resources refer to real-world objects and are outside of the Web's information space. They can not be dereferenced directly, wherefore the requested server that hosts a non-information resource returns a HTTP response with status code `303 See Other` [FGM+99]. This process is called a *303 Redirect* (cf. [BCH07]). The server's response contains the URI of an information resource that describes the requested non-information resource in more detail; it provides a machine-processable representation of the requested non-information resource.

Despite the simplicity of the underlying conceptual model, RDF offers expressive power and flexibility while exposing a relatively generic triple-based model; by identifying concepts as well as the relationships among them using URIs, RDF documents can be linked together to form a larger data graph[29]. Using RDF as a representation language and storage model for contextual data offers a number of significant advantages compared to other representation and storage formats (cf. [Owe09, HRSZ11]):

- *Data aggregation*: due to the simple yet flexible underlying conceptual schema, aggregating RDF data can be realized by specifying relationships between resources from different data sources or RDF repositories, which can be uniquely identified and referred to by their URIs. In this way, data sources can be linked together by adding new RDF triples that express relationships between related resources or allow for defining mappings and rules for schema realignments (more expressive languages such as OWL allow for specifying more complex and logic-based relationships such as functional, reflexive, transitive, inverse properties etc.), which is a more viable form compared to the complex schema realignments in relational database management systems [Owe09]. Additionally, if two resources are using the same URI, its valid to assume that they refer to the same (real world) entity.
- *Data discovery*: by using URIs for identifying resources and concepts, data discovery across documents and systems is facilitated since the use of URIs allow for a global identification mechanism among systems and domains. Identical resources can be discovered among different RDF documents in case they use identical URIs to refer to information and non-information resources (e.g. `http://www.dbpedia.org/resource/Vienna` to refer to the city Vienna and `http://www.dbpedia.org/data/Vienna.xml` to refer to a information resource that contains information about Vienna[30]). It is common Web practice to assume that identical URIs conceptually refer to the same resource and denote identical concepts.

[29] Please note that the notions of RDF document and RDF graph are used synonymously across this section for simplicity reasons although a graph refers to a more generic and mathematical notion whereas the notion of RDF document is used to emphasize the RDF vocabulary classes and instances (also denoted as resources or individuals).

[30] In this respect, the information resource `http://www.dbpedia.org/data/Vienna.xml` is a representation of the non-information resource `http://www.dbpedia.org/resource/Vienna`

- *Data representation*: URIs exhibit a standard and uniform identification scheme that is used to identify and refer to the elements defined in a controlled vocabulary to represent the knowledge of a particular domain – the universe of discourse. Since URIs are used to identify such elements, elements from different vocabularies and ontologies can be exchanged and utilized among different RDF documents and work in a networked way. By making the semantics of such elements explicit, different RDF documents using the same controlled vocabulary for describing their data can be interpreted and processed identically in different systems. Since RDF-based vocabularies are built on higher-level languages such as RDF/S and OWL that were defined on formal and well-defined logics, knowledge not explicitly asserted in an RDF document can be deduced by drawing logical conclusions on existing assertions. Additionally, RDF supports the concept of *reification* [HM04] and facilitates the reconciliation of structural heterogeneity due to its generic triple-based data model.

- *Data flexibility*: RDF is an open world framework with no tightly defined data schemas, the underlying data model conceptually does not impose any limitations or restrictions (besides complying with its triple-based structure) to an RDF graph in terms of the data being represented or the amount of data to be contained in a graph. RDF offers a great flexibility for expressing almost any kind of information where *anyone can say anything about any topic*. This allows to add data to an RDF graph even if the structure is not known beforehand, which is a significant advantage compared to database management systems that rely on tightly defined database schemas [TGE+06, Owe09].

- *Tool support*: RDF offers a powerful and open knowledge representation language that can be used to virtually express any type of knowledge due to its underlying generic triple-based schema. This simplicity and flexibility led to the creation of a multitude of open-source frameworks and tools (e.g., knowledge extraction engines, reasoners etc.) for processing and storing RDF data that can be used in existing applications.

In the following paragraphs, we elaborate on the benefits resulting from the usage of RDF/S and ontology-based vocabularies for representing contextual information compared to hierarchical XML-based context representations. However, the benefits of using RDF instead of XML as a description framework have been discussed since the emergence of RDF itself [BL98].

Generally, the logical structure behind an XML document can be expressed in multiple ways, which on the other hand influences query languages that should be ideally independent from a concrete representation. Since RDF is based on a simple directed triple-based graph, the logical structure is consistent among every RDF document. By mapping XML documents to semantic graphs, one must consider the one-to-many relation that exists between a semantic graph and the various XML representations since an RDF graph can be represented using a multitude of different hierarchical XML graphs. Additionally, the underlying schema must be known beforehand to elaborate on the structure and semantics expressed in an XML document. Due to the different possibilities of representing identical facts in XML, querying for information is more complicated in an XML tree in terms of the different structural possibilities an XML document incorporates. This complexity increases for combined queries, that is, queries containing more than one property. Moreover, traditional XML schema definitions such as DTDs lack an inferencing support and inferencing language [BL98].

By representing context descriptions as RDF graphs, a context consumer can process the triples it is interested in and ignore irrelevant ones. XML in contrast requires a schema and explicitly

specified semantics (mostly encoded in applications) to interpret and infer on the semantic relationships between the different context items and properties. For interpreting the semantics of a context description as well as the semantics of the relationships between context elements, an XML parser and the application making use of it respectively have to interpret the structure and schema information of the XML document. In RDF, the semantics of context elements can be specified explicitly and independent of a document's structure as opposed to XML, where the semantics of a document is implicitly encoded in the structure and composition of its constituting elements.

Another problem is that XML schemas can not be extended without influencing query expressions, that is, query expressions must be adapted according to changed schemas. RDF in contrast allows to add new data to context descriptions without the necessity to adapt existing query expressions, which can be applied without modification. Ambiguities in context descriptions can also more easily be handled and reconciled using RDF since the semantics of asserted statements are defined externally to the document in which they were used. When new facts are introduced to XML-based context descriptions, they need to be compliant to the prescribing schema.

2.7.1 Representing Contextual Aspects

Context in general represents an information space that can be well represented using a *directed labeled graph*, which is the underlying data model of the Semantic Web languages RDF and OWL (cf. [EPR08]). A node in an RDF graph refers to the resource identifying a *contextual aspect* such as location or temperature where more complex contextual aspects may be represented by sub graphs consisting of several nodes and edges. Edges represent relationships between elements describing a contextual aspect where the semantics of a relationship is determined by the vocabulary term used to describe a relation and the direction of an edge. Two nodes connected by an edge form an *RDF statement* or *RDF triple* (cf. Definition 2.2) and represent an elemental assertion about a contextual aspect or a part of it. An edge in an RDF graph is equivalent to a predicate of an RDF statement and constitute a specific *context property* where the object of a statement may represent a specific *context property value* being a plain or typed literal (cf. Section 2.7.3), refer to another resource, or represent a structured value using the concept of *blank nodes* [KC04]. Based on these conventions, a contextual aspect is defined as follows:

> **Definition 2.5** (Contextual Aspect)**.** *A contextual aspect is a relevant and distinct part of the surrounding real-world context such as the current location or the current temperature (cf. [RSP07]) and is identified using a concrete URI. A resource and associated statements represent a concrete instantiation of a contextual aspect and describe the elements being relevant to it.*

An instance of a contextual aspect thus comprises all the RDF statements[31] that adhere to the resource representing a contextual aspect and make assertions about the elements being relevant to it; for instance, the contextual aspect of the user's current location can be described using the statements depicted in Figure 5.4 on page 181. Moreover, contextual aspects that are logically or semantically related together form a *contextual constellation* that represents an constituting part of the user's real-world context.

[31] We use the term 'triple' and 'statement' synonymously throughout this chapter.

In general, we suggest to make use of the concept of blank nodes [KC04] for the description of contextual aspects, in particular when there is no need for globally identifying a specific resource or for expressing complex contextual aspects respectively contextual constellations. For instance, the use of anonymous nodes are general means to express restrictions on concept and property instance values in OWL (using `owl:Restricition`) such as `owl:AllValuesFrom`, `owl:SomeValuesFrom`, or `owl:hasValue` [AH08]. Moreover, RDF only supports binary relationships per default; by validating a path expression we can asses whether a given relation holds between two resources (given that the object is not a literal). However, blank nodes allow to represent *n-ary relationships* between a specific context-relevant resource and its constituting elements.

Different contextual aspects can be classified by means of a common concept using the `rdf:type` property that indicates the correspondence of an instantiation of a contextual aspect to a specific classification class. According to the RDF semantics, the class specified as a statement's object represents the category or class from which the corresponding subject is an instance of[32]. Resources that are instances of concrete classes are called *typed nodes* in terms of RDF graph semantics or *typed node elements* in terms of RDF syntax [Bec04].

2.7.2 Identifying Contextual Information

To identify contextual information and contextual aspects respectively, a common approach is to make use of *fragment identifiers*[33] [BL97]. A fragment is interpreted relatively to a given *base URI* and allows to represent different aspects as well as different revisions of a user's context. In the context of the Semantic Web, fragment identifiers are commonly used for referring to vocabulary terms or to concepts of an ontology as illustrated in the subsequently given example, in which the URI of the concept `Context` from the data description ontology is presented.

```
1    http://www.datadescription.org/2010/09/vocabulary#Context
```

By using the base URI plus the fragment identifier, a context consumer can request a particular contextual aspect it is interested in by adding the fragment identifier to the base URI of the context description hosting an instance of a contextual aspect. By assigning a fixed URI to context descriptions created by a context framework, these can be requested regardless of their individual identifiers by de-referencing the respective static URI (cf. [Lew07]).

2.7.3 Representing Context Property Values

For expressing concrete context property values, RDF offers the concept of *typed literals* [KC04] to represent such data as structured values in order to standardize processing and interpretation tasks among context consumers. In contrast to *plain literals* which are untyped and self-denoting literals (cf. [KC04] Section 6.5), typed literals exhibit datatype-specific information and allow for expressing context-specific numerical values in a structured and controlled way. By using typed literals, the interpretation of context property values adhering to a contextual aspect is not restricted to the applications or services in which a context description is consumed but explicitly represented in it. In [KC04] Section 3.4, typed literals are defined as follows:

[32] Please note that according to the RDF semantic, a resource can be instance of multiple classes (cf. [MM04]).
[33] However, the term 'fragment' is misleading as a fragment identifier can identify anything (not only document fragments) [BL97].

Definition 2.6 (Typed Literal)**.** *A typed literal is a string combined with a datatype URI. It denotes the member of the identified datatype's value space obtained by applying the lexical-to-value mapping to the literal string.*

Typed literals in RDF consist of a prefix string and an associated URI reference that identifies a particular XML Schema data type [BM04]. The concept of typed literals can be used for representing time stamps that refer to the creation date of a context description, which is important for their synchronization. For instance, the following fragment of an RDF-based context description illustrates the use of typed literals for explicitly representing synchronization-relevant information (indicated by the `context:timestamp` property) using the XML Schema datatype `#date`[34]:

```
<urn:uuid:baac630a-5cdb-4c79-92e6-6ce3d07419bc>
   a context:Context ;
   context:timestamp "2009-06-16T15:58:22"^^<http://www.w3.org/2001/XMLSchema#date> ;
   context:previous <urn:uuid:d3ee316b-5704-4893-acb9-df1495c79011> .
```

By explicitly modeling such information, a context framework or context consumer can infer whether a requested context description has been recently created and reflects the user's current context or whether it refers to a previous context[35].

Due to the fact that RDF does not incorporate a dedicated data type system that defines a value and lexical space for representing data types nor provides modeling primitives for explicitly defining data types (apart from the built-in data type `rdf:XMLLiteral` itself), RDF makes use of XML Schema data types [BM04] to align an XML Schema data type such as `xsd:integer` or `xsd:date` to a literal in order to provide explicit information regarding its interpretation. This information concerns the *value space*[36] that the data type represents, the *lexical space*[37] a data type must adhere to, as well as a *lexical-to-value mapping*. To illustrate this exemplarily, the value space, lexical space, and the lexical-to-value mappings for the XML Schema data type `xsd:boolean` are depicted in Figure 2.8[38]. An XML Schema data type assigned to a literal can be identified by its specific XML Schema URI and is thus being defined externally to an RDF document (see [MM04]). The inclusion of standard XML Schema data types facilitates the interoperability between context frameworks and context-aware applications since shared contextual information and the included context property values can be interpreted externally to the application logic.

As RDF does not inherently define data types, the interpretation of typed literals is left to applications that process an RDF document. Hence, an application is responsible for determining the validity of assigned XML Schema data types to literals, e.g., whether or not the value of a typed literal is in the lexical space of the corresponding XML Schema data type. Moreover, by using external schemas for identifying specific data types, the interpretation as well as determination of the correctness and validity of a specific data type is also part of the respective application. By using typed literals whose URIs refer to XML Schema data types, a context producer is able to make explicit assertions on how to interpret given context property values which in turn facilitates the exchange of contextual information.

[34] Please note that data types are defined externally to RDF and are referred to by their *URIrefs* (e.g., the URIref for the data type `xsd:integer` is `http://www.w3.org/2001/XMLSchema#integer`).

[35] Supposing that the timing has been synchronized beforehand.

[36] A value space is defined as *"the set of values for a given datatype"* where *"each value in the value space of a datatype is denoted by one or more literals in its lexical space"* [BM04].

[37] The lexical space determines the set of literals that are valid for the data type it was defined for.

[38] The lexical-to-value mapping for the XML Schema datatype `xsd:boolean` has been compiled from [KC04].

```
Value Space:              {T, F}
Lexical Space:            {"0", "1", "true", "false"}
Lexical-to-Value Mapping: {<"true", T>, <"1", T>, <"0", F>, <"false", F>}

Typed Literal            Lexical-to-Value Mapping    Value
-------------------------------------------------------------
<xsd:boolean, "true">    <"true",  T>                  T
<xsd:boolean, "1">       <"1",     T>                  T
<xsd:boolean, "false">   <"false", F>                  F
<xsd:boolean, "0">       <"0",     F>                  F
```

FIGURE 2.8: Value space, lexical space, and lexical-to-value mapping for the XML Schema data type `xsd:boolean` and their typed literal definition

2.7.4 Using Structured Properties for Representing Contextual Information

In some situations, typed literal fail to provide the necessary expressiveness to explicitly specify the interpretation of complex context property values. Typed literals presume the existence of implicit unit information for a stated value. This might be adequate for certain types of information such as location-based coordinates (e.g. the latitude or longitude coordinates could be modeled in RDF as `geo:lat "48.175443"` and `geo:long "16.375493"`) but fails for other information types, in particular for physical or cultural-dependent information units such as temperature, weight, dimension, or price, which require explicit units or measurement information. For those cases, RDF includes the concept of `rdf:value`-properties that allow for explicitly specifying the "context", i.e., the unit of an acquired value. Hence, the value of a context property is represented as a blank node [KC04] being the subject of additional statements that describe how the value is to be interpreted. Structured values are usually represented by adding a separate blank node that has the `rdf:value` as well as the qualified property value attached to it, plus additional unit or measurement properties and corresponding values (in most cases this is the URI of the concept representing the unit). Figure 2.9 exemplifies the use of structured property values for representing temperature values according to different temperature scales:.

```
[] a context:Sensor ;
   exterm:temperature [
      rdf:value "24.6"^^xsd:decimal ;
      exterm:unit exunit:Celsius rdfs:label "Temperature in Celsuis degree" .
      rdf:value "76.28"^^xsd:decimal ;
      exterm:unit exunit:Fahrenheit rdfs:label "Temperature in Fahrenheit" .
   ] .
```

FIGURE 2.9: Use of `rdf:value` property for representing the values of a temperature sensor

By attaching unit information to context property values, a context description may host different values referring to a contextual aspect. In case a context consumer specifies that it requires context information to be measured in a particular unit, a context framework can request a translation service and adapt context property values to the requirements stipulated by the requesting client. In doing so, a context description can be enriched with alternative values regarding its acquired context property values which allows for adding structured information from additional classification schemes in order to further describe a given context property value.

2.8 Semantic Web-enhanced Context-aware Computing

For managing context information systematically, a common structure for representing contextual information need to be established [KMK+03]. Since technologies and concepts from the Semantic Web have been designed for heterogenous environments, they offer languages and technologies that serve as standards for expressing contextual information, and can therefore be shared and exchanged among systems and applications. The Resource Description Framework, discussed in the previous section, has proven to be an appropriate representation framework for representing complex contextual constellations and facilitating the sharing and exchange of context descriptions based on ontological semantics [ZS10]. It can be used for codifying the semantics of contextual information as well as the relationships among them in a well-defined, uniform, and systematic way. Its open architecture allows for the integration of different vocabularies to describe contextual information so that context descriptions can dynamically grow and become more elaborated. Additionally, RDF allows to represent contextual information in multiple ways so that such information can be utilized by a wide variety of context consumers. On top of RDF, RDF Schema (RDFS) offers a simple set of common language properties that can be used for building context descriptions which can be shared among different context providers and consumers collaboratively [KMK+03]. However, the set of RDFS language elements is not sufficient for expressing rich contextual constellations, wherefore the use of more expressive languages such as OWL is suggested [MT07, EPR08]. Generally, the use of ontologies as a key component for building a context-aware computing framework is broadly acknowledged [KMK+03, PdBW+04, EPR08, LFWK08], and it has been shown that Semantic Web technologies are sufficiently mature and performant to be deployed on mobile devices [ZS11, ZS12b].

Several works have already demonstrated that ontologies are appropriate means for expressing and representing contextual information since they incorporate some characteristics that are essential for mobile and ubiquitous environments [CJ04, HDM05, HMD05, EPR08]. Ontologies are highly expressive and widely adopted knowledge representation techniques, and a multitude of open software tools for their design, creation, management, and storage are available [PvHS07]. Ontologies offer a well-defined set of concepts and relationships to model the domain of interest, which can be adopted by context-management frameworks to integrate and share contextual knowledge from other domains to facilitate context exchange and reuse. Ontologies are based on knowledge representation languages that are open with respect to evolutions of the domain they describe. This allows ontologies to be adapted and extended according to domain-internal changes. The use of ontologies as languages for representing contextual information is further supported through the availability of tools that helps in designing and managing ontological vocabularies. Since most of the technologies and languages found in the Semantic Web were defined on the basis of open standards, this fact further lowers the barriers for their adoption in distributed and heterogeneous environments.

Semantic Web languages such as RDF and OWL were specifically designed for highly distributed and heterogeneous environments and use *HTTP URIs* [BL02] to reference both information and non-information resources (cf. [BL05, BCH07]). URIs do not distinguish between locally or remotely hosted resources. Due to the single global address space of Web URIs, naming conflicts between different context representation languages can be easily reconciled. Especially when adding new context acquisition components, which were not available at design time, ontologies allow for information transformations and matching even if the information does not exactly match information requirements [EPR08]. Due to the fact that URIs use a global address space, Semantic Web-based vocabularies and ontologies used for describing contextual data can work in a networked way.

Building a context awareness computing infrastructure on the principles and technologies of the Semantic Web has several implications and advantages: due to the fact that ontologies are based on the *open world assumption*, context ontology evolution is a central aspect in context management and allows for adapting and modifying a context ontology according to changed conditions. This makes them applicable to dynamic, unpredictable, and frequently changing environments. Due to the fact that RDF is a system- and application-independent framework for modeling data and its close relatedness to Web technologies, it is well suited for the data exchange between components (e.g., using HTTP). This facilitates interoperability among context frameworks and services since established and well-known vocabularies together with their inherent semantics can be understood and used across systems.

Since ontologies provide a common structure for representing and describing the relationships and semantics of context-relevant information in a machine-processable way, they can be conceived as a general approach for systematic management of context information. In such a setting, RDF can be used as a description syntax to enable the communication and sharing of context information between collaboratively communicating partners, i.e., applications, services, and devices. These descriptions are represented as *labeled multi graphs*, where the contained entities are referred to through HTTP URIs (see Section 2.6). Its open architecture allows for the integration of different context-relevant vocabularies so that context descriptions can dynamically grow and become more elaborated to better reflect intra-domain evolutions.

Additionally, ontologies facilitate the interpretation of sensed or derived values to allow for their aggregation and transformation into symbolic values, i.e., transforming collected data into statements adhering to a prescribed vocabulary. Hence, context acquisition components do not need to anticipate possible queries beforehand, but provide the data they have and let the requesting components decide which information is of relevance to them. Additionally, if context-relevant data are represented using Semantic Web languages, they can be integrated and processed even if they were not known at design time of a mobile system (see [EPR08] page 30 for an example). This also applies to divergent sensor or service feature descriptions where the identification of correspondences between heterogeneous descriptions serves as a basis for utilizing services and integrating acquired information that had not been anticipated at design time of a mobile system.

Ontologies help in expressing application or service needs, and in aligning them to acquired context information wherefore only relevant information is extracted. This simplifies query processing since a context consumer can limit queries to relevant information, instead of processing the entire context description. In cases of incompatibility of context descriptions, ontology matching algorithms help in reconciling differences in description semantics. Euzenat [Euz05] therefore suggests the use of *ontology mediators* or *ontology alignment services*[39] to identify correspondences between incompatible context descriptions. Such services help in identifying correspondences between different context models used by context producers and consumers, perform query transformations, and reflect domain and information space evolutions [EPR08]. Other works such as [WX06] suggest to use a dedicated model matching component as part of the infrastructure in which it is deployed. If a sensor interface is expressed using an RDF vocabulary, such services can be used for aligning and translating the exposed sensor information to the ontologies used by a context framework. However, this aspect will not be addressed in the course of this thesis and is subject of further research.

[39] An ontology alignment service is a sub-domain or field of ontology matching whose result is a set of correspondences (the alignment between ontologies) that can be used for merging ontologies or transforming queries issued against a model [EPR08].

Semantic Web technologies facilitate both direct and indirect context awareness, since context-related information can be acquired from external services or repositories in a structured and well-defined way based on explicitly represented semantics using open standards. Those technologies allow for mapping low-level sensor data to high-level ontological concepts so that collected context-relevant information is transformed and embedded in a controlled context description. Based on ontological semantics, new facts can be deducted by applying aggregation and reasoning heuristics. In this way, Semantic Web languages such as RDFS and OWL allow for aggregating heterogeneous and autonomously acquired context information both on a syntactic and semantic layer. By transforming sensorial data into RDF statements, context acquisition components are not required to anticipate possible queries beforehand. Instead, the requesting context consumers determine the data that are relevant for them.

Additionally, Semantic Web technologies can further be used to describe the source from which contextual data were gathered (cf. data provenance) by using meta data or annotations in order to indicate the *trustworthiness*, *reliability*, and *stability* of a context source. Context information with a high trustworthiness factor could be preferred by context reasoners to reduce the computational resources involved and assessing the truth-value of a contextual fact [HMD05]. The specification of *meta-information* allows a framework to facilitate the integration of contextual information acquired from remote sensors by attaching meta-information to acquired contexts and sensors to make additional assertions wether such data are contradictory, inaccurate, or incomplete. This aspect is also often referred to as *Quality of Context (QoC)* where predefined parameters help in assessing the trustworthiness according to the requirements imposed by the underlying framework and the applications operating on it (cf. [BS03, BTC06, MTD08]).

In the fields of Semantic Web and Linked Data, a number of vocabularies and ontologies have emerged that are of interest for the representation of contextual and situational information (e.g., time[40] and location[41], technical parameters[42], or social aspects[43]). The elements (terms and concepts) of those vocabularies are well-known across communities and expose a well-defined and commonly understood semantics that allows for information integration and exchanges especially in heterogeneous system and network infrastructures. Additionally, such vocabularies are continuously maintained by communities to guarantee their accurateness and evolution. By re-using such vocabularies and (implicitly) connecting context descriptions to external Linked Data sources, we gain two benefits: first, if context descriptions are distributed (either to the public or within a closed environment, e.g., a corporate network) they can be directly combined with already existing data, and existing tools can be directly applied to contextual information without the need to adapt existing software. Second, data from external sources can be imported and used to enrich the context descriptions, leading to a richer semantics, which facilitates more powerful processing and reasoning. Not being bound to a single vocabulary also adheres to the idea of dynamic and flexible context descriptions evolving in the course of user-relevant activities that can not be determined a priori – especially not at design time of a mobile system or mobile application.

The flexibility and generic schema of the RDF data model combined with the usage of URIs as identification and addressing scheme allows for the combination of different vocabularies in an RDF document, which is common practice in many Semantic Web applications. Therefore, it is good practice to not define new vocabularies from scratch but to use existing vocabularies and extend them with the concepts and properties needed [BCH07]. Context descriptions can benefit

[40] OWL Time Ontology: `http://www.w3.org/TR/owl-time`
[41] Basic Geo (WGS84 lat/long) Vocabulary: `http://www.w3.org/2003/01/geo`
[42] Composite Capabilities/Preference Profiles: `http://www.w3.org/Mobile/CCPP`
[43] The Friend of a Friend (FOAF) project: `http://www.foaf-project.org`

in two ways when contextual information is expressed using terms from well-defined semantic vocabularies and linked data sources: (1) linked data URIs are *dereferenceable* (cf. [Lew07]), i.e., when a contextual aspect is identified using an URI from a linked data source, a description can be retrieved for it, and (2) linked data resources are already linked to other data sources in many ways so that elements from context description can be easily reconciled [HB11]. Although, RDF-based descriptions are intended for machine consumption, it is nonetheless useful to add human-readable information such as explanation for the defined terms. For a discussion about good practices for publishing RDF vocabularies on the Web, the reader is requested to consult [BPM+08].

The notions of context and context awareness are also of importance for the Semantic Web, especially for ontology alignment and ontology-based information integration since ontologies—according to their definition as shared conceptualizations of a domain (cf. [Gru95])—expose an implicit conceptualization of the universe of discourse they formalize and require the notion of context to elaborate on and transform the claims made [Bon04]. Context can support this process as it helps to specify the situations and circumstances in which certain claims had been made. In this respect, context is used for codifying provenance information that helps in ascertaining the trustworthiness and validity of context-related data.

2.9 Discussion

For leveraging the full potential and opportunities of mobile information systems and devices, context awareness need to be considered a central aspect of mobile computing [Teo08]. However, current approaches for integrating context awareness in mobile systems fail to consider the dynamic and ambiguous nature of context, which require a holistic treatment of context and context awareness taking into consideration the manifold and versatile nature of context types as well as the relationships among them [KA04]. One reason for that is that static and external factors are easy to capture since they adhere very well to existing software methodologies [Teo08]. Soft factors such as the user's intentions, believes, preferences, information needs, goals etc can not be sensed or inferred indirectly. Mobile information systems are likely to benefit most from context-aware computing if the notion of context awareness moves from a *system-centric* towards a *user-centric* view since context awareness is an activity-driven process focusing on users' activities and information needs [CX06, SLGH08]. A user-centric view on context awareness leads to a general model of context-aware mobile computing that is more intuitive, productive, and consistent according to the user experience [Teo08].

Especially in the information systems discipline, context is considered as a representational issue where the focus is put on its codification and representation. According to this conception, context can be scoped in advance, is independent from its instantiation, and can be separated from user activities. This renders the determination of a set of relevant contexts or contextual aspects in advance difficult, if not impossible [Gre01]. The static conception fails to consider context as a dynamic, emergent construct that is formed by interaction and can therefore not be determined a priori, and it is unable to provide a holistic view or treatment of context awareness in which the user's role, her intentions, and her activities are central aspects. Instead, context should be conceived as a phenomenological construct whose scope is defined dynamically and in an instance-dependent way since context emerges opportunistically in the course of interaction and is being continually redefined and renegotiated; context is an emergent feature of interaction [Dou04] that differs among instances and is being actively created where an activity itself can be the context of another activity [Teo08]. Context should be considered as a dynamic and

emergent phenomenon that is relevant for a certain amount of time and whose scope is defined dynamically and unpredictably.

2.10 Summary

Our analysis affirmed that the way context and context awareness is conceived by people differs fundamentally from how it is computationally processed and utilized, and demands for a technical, conceptual, and methodological reconciliation (cf. [Eri02]). People usually interpret visual and auditory signals based on their previous experiences (their mental models) that helps them to define the context relevant for their actions. Context-aware systems in contrast are only able to capture a limited space of the current situation that is determined by their implemented sensors and lack the human intellect and common sense of conceiving a situation holistically and behave intelligently [Eri02, KPL+04]. Instead, the main purpose of context-aware computing is to serve for the automatization and autonomous execution of electronic tasks based on certain contextual constellations while shielding computational tasks from the necessity of explicit human attention and involvement [Eri02]. The objective of context-aware computing is not to create systems that adopt human behavior and react as humans would do in comparable situations, but to provide the technical and conceptual background that allows for creating applications and systems that can tailor their behavior and information retrieval processes according to specific contextual constellation.

Context is continually and ubiquitously redefined and negotiated in the process of communication and interaction. The central question in this respect is how to develop an accurate model of context and context awareness that continually adapts to unpredictable changes in the surrounding physical, virtual, and technical environment as well as on the changing information needs of mobile users. We believe that semantic technologies supplemented with reasoning and machine-learning technologies are appropriate means to facilitate the building of a context-ware infrastructure for mobile devices that continually adapts its replication processes according to current and future information needs.

In this section, we have outlined some of the areas in context-aware computing where concepts, technologies, and languages defined in the context of the Semantic Web can make substantial contributions regarding the representation, processing, and sharing of contextual information as well as in the reconciliation of heterogeneous context semantics. The potential benefits of semantic technologies for related areas such as pervasive computing have been discussed in previous works (e.g. [EPR08]). In the next chapter, we discuss relevant works that elaborate on mobile replication strategies, we analyze currently available mobile RDF frameworks and mobile RDF storage and persistence frameworks, as well as existing context-aware mobile Semantic Web applications that combine context-aware computing concepts with semantic technologies and ontology-based description frameworks. We then comprehensively introduce and define the constituting elements of our context-sensitive RDF data replication framework that implements the ideas and concepts presented in this section on a formal and conceptual basis. We denote this form of context-aware computing as *Semantic Web-based context-aware computing* [ZS12b].

Chapter 3

Related Work and State of the Art

"We shall not cease from exploration.
And the end of all our exploring will be to arrive where we started and know the place for the first time."

Thomas Stearns Eliot (1888 - 1965)

3.1 Introduction

In this chapter, we analyze existing works regarding the main building blocks of our work as outlined in the introductory chapter. For realizing our idea of making Semantic Web technologies available on mobile systems for the intelligent, context-dependent provision of user-related data, research into the current state of the art has been carried out along three directions: first we give a brief overview of mobile data replication and distinguish existing works from our approach; second, we analyze existing Semantic Web frameworks as well as current RDF query and storage infrastructures with respect to their appropriateness and deployability on mobile platforms; and third, we present and discuss existing Semantic Web projects that aim to synthesize semantic technologies, mobile systems, and context-aware computing.

3.2 Mobile Data Replication

The problem of replicating data to mobile devices is not new. Standard replication strategies—as known e.g., from relational data bases—cannot be directly applied to mobile scenarios because of the special restrictions imposed by changing context parameters, as outlined in Chapter 2. Therefore, several algorithms were proposed that estimate the costs of data usage based on various context parameters, and adapt the used replication strategies accordingly (e.g., costs of data transmission [HSW94], access frequency [WJH97], location [WW03], or device and environment characteristics [BGSA05]). These approaches are highly optimized towards single specific context parameters but do not consider the entire user context; especially they do not focus on the

semantics of replicated data. However, they can be considered complementary to our approach since they can be used to determine the frequency of replication updates.

Several approaches follow a more generic strategy and provide architectures that are extensible w.r.t. the considered context parameters and replicated data (e.g., [HS03, LBWK05]). However, all these approaches depend on a server infrastructure, on which context processing and inference tasks are performed. To the best of our knowledge, no approach exists that solely relies on processing executed on the mobile device itself, without depending on external components and services.

3.3 Semantic Web Frameworks for Mobile Platforms

Typical Semantic Web frameworks like *Sesame*[1], *Virtuoso*[2], *Jena*[3] [McB01], and *ROWLEX*[4], an open source toolkit developed by the NATO C3 Agency for the `C#` platform, hide the details of RDF data processing, serialization, and query execution from higher-level applications. However, these heavy-weight systems cannot be deployed on typical mobile devices because of their limited memory and processing capacities, latencies as well as incompatible application models and operating system infrastructures (cf. [FZ94, KLO$^+$04]). Those frameworks are usually developed for powerful server or desktop computing infrastructures incorporating many-core architectures, whereas mobile devices in general contain low-powered single-core RISC-based processors whose architecture was not designed for processing large data amounts.

Although such systems have proven to be powerful means to process, store, and reason over RDF data, they cannot be efficiently deployed on mobile systems due to the previously mentioned reasons and are therefore not considered in our related work analysis. Instead, we exclusively concentrate on RDF frameworks that have been specifically designed for deployment on mobile platforms and are available as Java libraries as well as on mobile query and storage frameworks that are built on top of existing RDF frameworks and provide additional functions for local RDF data query and persistence.

In the following, we give an overview of existing XML and RDF frameworks available for mobile platforms as well as query and persistence frameworks, where we include all publicly available solutions that have been referenced and described on the Web or were discussed in related publications throughout the past years (see [Zan09, HDW09, ZS11]). We categorize these works in *XML parsers* that were extended with RDF and OWL support, pure *mobile RDF frameworks*, and *query and persistence frameworks*, since pure RDF frameworks usually lack dedicated query and storage functionality [Zan09, ZS10, ZS11]. For each framework as well as query and storage infrastructure, we provide a short background description together with details regarding the following functional and non-functional features:

- *Licensing Model.* A license and in particular a software license usually defines the terms and rules that determine how to use a particular software or parts of it and grants permission to manipulate or adopt a piece of software. We ascertain the type of license under which

[1]Sesame: `http://www.openrdf.org`
[2]Virtuoso: `http://virtuoso.openlinksw.com`
[3]Jena: `http://jena.sourceforge.net`
[4]ROWLEX: `http://rowlex.nc3a.nato.int/`

a framework was released (e.g., EUPL[5], GNU GPL[6], Apache License[7], etc.) and whether a framework can be used in commercial and non-commercial applications.

- *Deployment and Installation.* Some frameworks are released as Java libraries (`.jar`-files) or as Android applications (`.apk`-files), which influence their deployment on a mobile device as well as their integration in existing applications. For instance, the integration of frameworks released as `.apk`-files in existing applications requires manual involvement and also puts restrictions on the utilization of exhibited functionality. For instance, to share data between two Android applications, dedicated Intents and Intent Filters as well as corresponding Content Providers need to be defined (cf. [RLMM09, Mei10, And10]). Additionally, some frameworks require the installation of additional 3rd-party libraries to be fully operational.

- *Quality of APIs.* This feature examines the quality of the respective frameworks' APIs and concerns aspects regarding the completeness of an API. For instance, we specifically analyze whether functions are defined that were not implemented or whether there are undocumented classes or methods. Moreover, we examine whether some sort of exception handling was implemented and to which extend exceptions are covered in case of improper function calls.

- *Internal data model.* The internal data model used for managing and storing RDF data allows for extrapolating on how a framework performs under specific circumstances and settings (for instance, a `HashMap`-based storage of RDF data yields fast access times but requires more storage and management overhead compared to, e.g., a `Vector`-based data structure). More specifically, we analyze how RDF triples are stored internally, i.e, which projections and transformation algorithms are internally used to resemble the triple-based structure of RDF using programming language-specific data types and data structures. This allows for drawing predictions regarding the efficiency of a chosen data structure w.r.t. access times, storage overhead, etc.

- *Platforms.* Some XML and RDF frameworks were originally developed for specific mobile platforms such as *CDC*[8], *CLDC*[9], or *MIDP*-based[10] operating environments. This fact might influence the potential platforms on which a framework can be deployed. Due to the incompatibilities between different mobile Java Virtual Machines (JVMs), it is not possible, for instance, to deploy a framework developed for the Android platform on a MIDP device. The compilation of a framework for a specific JVM exacerbates its deployment on other Java-based mobile devices that expose a different, incompatible virtual machine[11].

- *Supported Semantic Web Languages.* This aspect outlines whether a framework provides explicit support for Semantic Web languages such as RDF, RDFS, OWL Lite, OWL DL, OWL Full, and OWL 2. We ascertain, which Semantic Web languages are supported natively, for instance, through dedicated libraries or API methods and ontology classes.

[5]The European Union Public Licence (EUPL) `http://www.osor.eu/eupl`

[6]GNU General Public License v3: `http://www.gnu.org/licenses/gpl.html`

[7]Apache License Version 2.0 `http://www.apache.org/licenses/LICENSE-2.0.html`

[8]CDC is a framework specification for deploying and sharing mobile Java applications on hardware-constraint devices such as mobile devices or set-top boxes. It defines a basic set of libraries and virtual machine features that the underlying runtime environment must exhibit. Further information are available at: `http://www.oracle.com/technetwork/java/javame/tech/index-jsp-139293.html`

[9]Connected Limited Device Configuration: `http://java.sun.com/products/cldc/`

[10]Mobile Information Device Profile (MIDP): `http://www.oracle.com/technetwork/java/index-jsp-138820.html`

[11]Further information can be found at: `http://developers.sun.com/mobility/overview.html` - accessed 15th June 2011

- *Inferencing.* Inferencing allows for drawing logical conclusions on asserted RDF triples based on the logic and semantics defined for specific RDF, RDFS, OWL language constructs. Those languages allow to define statements and combinations of statements in an axiomatic way. There exist different forms of inferences (e.g., transitive closure and class membership inferencing [HBS08]) differing in their complexity as well as the sophistication and completeness of information to be deduced from existing assertions. Since inferencing is generally problematic on resource-constrained devices such as mobile phones, we ascertain which type of inferencing a specific framework supports or whether inferencing is supported at all.

- *Query support.* To date there exists a number of query languages proposed for querying RDF data (cf. [PG01]) such as RQL, RDQL, SeRQL, N3QL, RDFQ etc. where SPARQL proved to be the de-facto standard query language currently in status of a W3C Recommendation. We outline whether some of the query languages proposed for RDF data are supported natively or whether a framework uses proprietary mechanisms and methods to query for data contained in an RDF document.

- *Persistence.* This feature refers to a framework's capability to store RDF data permanently on a mobile device. We investigate which methods and infrastructures for RDF data storage are available and whether the storage of RDF data on a mobile device is supported at all. However, a common way to store RDF data locally is to serialize it in a specific serialization format and write it to a file-based output stream that writes an RDF document stream to a file object on the local file system. Another possibility is to define projections for storing RDF data in mobile database management systems.

- *Supported Serialization Formats.* Currently, there exist many formats for serializing RDF data such as N-Triple, N3, Turtle, RDF/XML, and specific RDF/XML abbreviation formats based on informal conventions such as RDF/XML-ABBREV (cf. [Bec04, MM04]) etc. We analyze which formats are supported by a framework for parsing and storing, i.e., serializing RDF data. Since each format imposes a different complexity regarding its syntactical, lexical, and semantic parsing, not all serialization formats are supported natively by a framework. We also distinguish between input serialization formats for parsing an RDF document and output serialization formats for writing an RDF document.

- *Documentation.* The successful deployment and utilization of an RDF framework requires a sufficient documentation wherefore we ascertain whether there is an "official" documentation available for a framework as well as the language in which it is written. We also give an overview of the potential locations and sources, e.g., whether the official documentation is supported or complemented by a wiki system or other online resources. Additionally, we ascertain whether the documentation reflects the latest development status, i.e., whether there are differences in the framework's version described in the documentation and the latest version available.

- *Source code availability.* In order to extend a framework's functionality or to apply code corrections and bug fixes, it is desirable to have a framework's source code available for a subversion-based check out or download. This aspect therefore indicates whether the source code of a framework is freely available and can be extended or modified on demand[12].

- *Extensibility and Add-ons.* Since most RDF frameworks only provide basic, i.e., rudimentary functions for RDF management and processing, external libraries for querying, storage,

[12] In most cases, the availability of a framework's source code depends on the license applied to it wherefore the aspects of license and source code availability are not disjunct and are considered in conjunction.

or reasoning should be available and complement a framework's functional spectrum. We investigate whether external libraries or APIs are available for a given framework, what functionalities they offer, and how they are to be integrated in existing projects.

- *Latest release and version.* This aspect records the month and year when the latest version of an RDF framework was released at the time of writing. We also provide the versioning scheme together with the most recent version nomenclature.

3.3.1 Mobile XML Parsers

kXML[13] is a lightweight open source XML pull parser that was specifically designed for constrained operating environments and mobile platforms such as Applets or Java ME-based mobile devices. It is based on the *Common XML Pull API*[14] and combines the advantages of XML DOM and SAX parsers for aligning XML processing routines to the structure of an XML document and, at the same time, providing instant access to parsed document elements. It was specifically designed to be used in CLDC and MIDP-based runtime environments. kXML is available as a Java library (`kxml2-2.3.0.jar`) and due to the compatibility between the *Dalvik Virtual Machine* exposed by the Android platform and the J2ME Java Virtual Machine, kXML can be deployed on Android devices as well. kXML uses a `Vector`-based data structure for storing XML nodes, i.e., all children of a specific XML node are stored in the corresponding `Vector`-instance of its parent node. Documentation about the parser is available on the official project page and in form of a Javadoc delivered with the project's sources and hosted online[15]. Additionally, a number of articles about kXML version 1.0 and 2.0 have been published on the Web[16,17]. However, development stalled in 2006 at version 2.3.0 although the developers announced a revised version including some architectural changes, which has not been officially released yet. Since kXML is a small and lightweight XML pull parser, capabilities for handling and processing RDF/S or OWL documents have not been implemented nor are available as external libraries. Due to the lack of fundamental RDF processing capabilities, kXML revealed to be inadequate to be considered as a framework for processing RDF data on a mobile device.

NanoXML for J2ME (+RDF/OWL)[18] is an open source J2ME port[19] of the original non-validating XML parser *NanoXML*[20] for Java, released under the *zlib/libpng License*[21], that has been extended with RDF and OWL support. It is dedicated to mobile environments such as CLDC and MIDP-based operating environments and offers convenience methods for navigating and retrieving data from RDF and OWL documents such as resources or property values. Basic information according to its usage and deployment are available on the nanoxml-j2me wiki page[22] as well as in the framework's Javadoc[23]. RDF data are stored and processing internally using a `Vector`-based data structure. NanoXML for J2ME is available as a single Java library and can be applied to existing projects by integrating the `nanoXML-j2me.jar`-file into the Java Build Path. NanoXML for J2ME incorporates only very basic RDF processing capabilities; its API does not define RDFS or OWL specific methods or classes nor does it elaborate on RDF/S and

[13] kXML 2 parser: `http://kxml.sourceforge.net/`
[14] Common XML Pull API: `http://xmlpull.org/`
[15] kXML Javadoc: `http://kxml.sourceforge.net/kxml2/javadoc/`
[16] See `http://www.devx.com/xml/Article/11773/0`
[17] See `http://www.ibm.com/developerworks/xml/tutorials/wi-kxml/section2.html`
[18] NanoXML for J2ME (+RDF/OWL) project: `http://nanoxml-j2me.sourceforge.net`
[19] Java Platform 2, Micro Edition (J2ME): `http://java.sun.com/javame/index.jsp`
[20] NanoXML project: `http://devkix.com/nanoxml.php?lang=en`
[21] zlib/libpng License: `http://www.opensource.org/licenses/Zlib`
[22] NanoXML for J2ME Wiki: `http://sourceforge.net/apps/trac/nanoxml-j2me/wiki`
[23] NanoXML for J2ME Javadoc: `http://nanoxml-j2me.sourceforge.net/javadoc/`

OWL semantics. Additionally, no form of rule-based reasoning or RDF/S-based inferencing is implemented in its latest release that also includes only very generic exception handling. In addition, NanoXML for J2ME provides only rudimentary query functionality and does not offer native support for SPARQL or other Semantic Web query languages. External libraries, e.g., for RDF querying, storage, and reasoning are not available either. NanoXML for J2ME is able to parse and process documents in RDF/XML serialization format; RDF/XML-ABBREV and other serialization formats are not supported. NanoXML for J2ME does not offer native methods for storing RDF data permanently on a mobile device. However, RDF documents can be serialized in the RDF/XML format and directed to a `FileOutputStreamWriter` to store RDF data in file objects on the device's local file system.

3.3.2 Mobile RDF Frameworks

Mobile RDF[24] is a Java-based open source implementation for the RDF data model, providing a simple and easy-to-use API for accessing and serializing RDF and OWL graphs. It is released under the Apache License 2.0 and was specifically designed for Java ME Personal Profile[25] and Connected Device Configuration (CDC) compliant devices, which is one of the main drawbacks of this framework since these application environments are only supported by a comparatively small amount of devices, namely those that employ a CDC-specific Java Virtual Machine (JVM). Most current and older J2ME-compliant devices deploy the more widely-used CLDC profile. Due to the compatibility between the Dalvik Virtual Machine and the J2ME-CDC Virtual Machine, the framework can also be deployed on Android devices. Mobile RDF provides specific packages for creating, parsing, and serializing RDF/S and OWL ontologies, and supports RDF Schema type and property propagation rules as well as RDFS and rule-based inferencing. The framework's class architecture offers abstract classes and interface definitions for extending the default RDF type and property propagation rule implementations and integrating individually defined inference rules. However, RDF graph modifications like deleting or editing RDF triples are not supported in the latest framework release. The RDF/XML format is the only serialization format supported by default and the framework also lacks native SPARQL or other query language support. The internal selection-based query implementation requires explicit knowledge about the triples/statements to be searched for. Documentation for MobileRDF is available as Javadoc delivered with the project's source code as well as on the official project page that also includes few implementation examples and information about a prototypical implementation of a mobile client for fetching RSS feeds. Mobile RDF is available as a Java library and can be applied to existing projects by integrating the `mobilerdf-0.3.jar`-file into the Java Build Path. Mobile RDF does not offer any additional APIs, although dedicated packages for RDF graph rendering based on the Graph Modeling Language (GML) [Him] are integrated. Mobile RDF uses a `HashMap`-based data model for the in-memory management of RDF data where each statement is stored in a separate entry. However, it does not offer methods for native RDF data persistence; instead, RDF data can be serialized in the RDF/XML format and written to the mobile device's local file system. Mobile RDF provides implementation support for selected XML Schema data types[26]. It offers an individual XML parser implementation that is specifically optimized for parsing RDF/XML-based documents and contains a set of basic exceptions for RDF and OWL parsing errors.

[24] Mobile RDF project: `http://www.hedenus.de/rdf/index.html`
[25] Java ME Personal Profile: `http://java.sun.com/products/personalprofile/`
[26] See `http://www.hedenus.de/rdf/datatypes.html` for a list of natively supported XML Schema data types.

μJena[27] is an open source J2ME port of the popular Jena Semantic Web framework, targeted for low-capacity mobile and embedded devices and specifically developed for CLDC-1.1 and MIDP-2.0 platforms[28]. It was released in 2008 under the GNU General Public License. Although its API is currently in a prototypical state and only allows for processing RDF data serialized in N-Triples format, it covers the entire set of RDF and RDFS modeling primitives, provides ontology, and limited inference support, as well as convenience classes for handling OWL ontologies. Like in Jena, RDF data are represented on two levels: on the lower, generic graph-based level, μJena stores RDF data as triple nodes, where an RDF-based model API is deployed on top that offers convenience methods for accessing and manipulating RDF models. Additionally, it defines a number of exceptions for indicating parsing problems and handling processing errors. RDF data are stored and processed internally using a `Vector`-based data structure. Most of μJena's documentation is available as Javadoc generated from its source code; additional information regarding its implementation and usage has been published in form of a master thesis written in Italian. Since μJena is a Jena port, a large part of the official Jena documentation also applies to it. μJena is available as a Java library together with a Java Application Descriptor (`microJena.jad`-file) that contains deployment information of the μJena main library on MIDP-based operating environments. It can be applied to existing projects by integrating the `microJena.jar`-file into the Java Build Path. Like Jena, μJena does not offer native support for SPARQL or other Semantic Web query languages; instead, it contains specific `Selector` and `Iterator` implementations for querying RDF models internally and provides at least a basic set of query functionality. However, additional libraries, e.g., for RDF querying, storage, and reasoning, as well as other external libraries are not available for μJena. Additionally, it does not offer native methods for storing RDF data permanently on a mobile device although RDF models can be serialized in the N-Triples format and directed to a `FileOutputStreamWriter` to store it into a file object on a device's local file system. μJena includes a basic reasoner that computes a set of simple entailments such as transitive closures on sub-property and sub-class hierarchies; however, rule-based inferencing is not supported natively.

Androjena[29] is the most recent Jena port specifically created for the Android platform and was released in 2010 under the Apache License 2.0 as an open source project. It is built on Jena version 2.6.2 and includes adapted versions of the entire set of functions and libraries Jena exhibits such as full RDF/S and ontology support, inferencing, as well as reading and writing RDF data in different serialization formats. Androjena includes specific reasoner implementations that support forward and backward chaining as well as rule-based reasoning. Just as Jena, Androjena is capable of processing RDF documents serialized in RDF/XML, RDF/XML-ABBREV, N-Triple, N3 and the Turtle format[30]. The Androjena core libraries do not include specific APIs for querying RDF data, local persistence, Named Graphs [CBHS05], or support for external reasoners. However, to provide at least a minimum of query functionality, the Androjena project page also hosts the *ARQoid* project[31], which is a reduced port of Jena's SPARQL query engine ARQ[32] and the *TDBoid* project[33] which offers a non-transactional, java-based, persistent storage and query infrastructure for Android. Currently, TDBoid and ARQoid are in prototypical status where ARQoid lacks some of ARQ's original features such as full-text query support. Currently, Androjena does not deliver a Javadoc for its libraries but hosts a Wiki on its official web page as

[27] μJena project page: `http://poseidon.elet.polimi.it/ca/?page_id=59`
[28] See `http://poseidon.ws.dei.polimi.it/ca/wp-content/uploads/install.txt`
[29] Androjena project: `http://code.google.com/p/androjena/`
[30] There exists different sub formats and sub writers for the N3/Turtle format such as a pretty printer, a record/frame-oriented format printer, and a prefix-based printer.
[31] ARQoid: `http://code.google.com/p/androjena/wiki/ARQoid`
[32] ARQ: `http://jena.sourceforge.net/ARQ/`
[33] TDBoid: `http://code.google.com/p/androjena/downloads/detail?name=tdboid_0.4.zip&can=2&q=`

well as a discussion group[34]. Since Androjena is an official port of the Jena framework for Android, the official Jena documentation[35] also applies for Androjena. Androjena is available as a Java library (`androjena_0.5.jar`) plus a set of required libraries that host additional functions Androjena requires such as Unicode (`icu4j-3.4.5`) and URI creation (`iri-0.8.jar`) support. Androjena exhibits a proprietary implementation of a data structure called `FasterTripleStore` for handling and processing RDF data internally, which resembles concepts from a native triple store implementation that stores subjects, predicates, and objects independently in dedicated `NodeToTripleMap`-instances. As outlined, Androjena does not provide dedicated or native functions for local RDF data persistence; instead, RDF models can be serialized in different formats and directed to a `FileOutputStreamWriter` to store it in a file object on a mobile device's local file system.

3.3.3 Query and Persistence Frameworks

RDF On the Go[36] is a full-fledged RDF storage and query framework specifically designed and implemented for mobile devices that was released under the new *BSD 2-Clause License*[37] and features the Android operating system. It follows an approach similar to Androjena, as the Jena core APIs together with the ARQ library have been adapted to the Android platform to allow developers to directly operate on and manipulate RDF data models. The primary storage infrastructure are *B-Trees* as provided by a lightweight version of the *Berkeley DB*[38] adopted for mobile usage and deployment. The internal query processor provides support for both standard and spatial SPARQL queries, where an *R-Tree* based indexing mechanism is used for storing URIs with spatial properties [LPPRH10]. The current version as of March 2011 supports a large set of standard SPARQL query operations where aggregation, sorting, and some spatial operations are subject to future implementations [LPPRH10]. RDF On the Go is available as a Google Android application (`.apk`-file) and need to be installed manually on a device[39]. Information about the RDF On the Go project can be found on the official wiki[40] as well as in a separate conference paper [LPPRH10]. A number of projects are hosted within the project's repository but further information whether these correspond to RDF On the Go could not be found[41]. Since the project's source code has not been released yet, further details regarding the implemented internal data model as well as supported Semantic Web languages, and inferencing technologies could not be provided.

SWIP: Semantic Web in the Pocket[42] was developed in order to support RDF data storage and exchange in a uniform, schema-less, and system-wide way based on the Linked Data principles [BHBL08]. SWIP represents an Android-specific implementation of an RDF storage infrastructure that is based on the Android-internal concept of *ContentProviders*[43] for application-wide data storage and exchange across applications and processes. It maps URIs to data stored in the local SQLite database deployed on Android systems and returns data in the form of triple sets or tuple tables. It employs a simple subject-predicate-object table layout for RDF data

[34] Discussion group for Androjena: `http://groups.google.com/group/androjena?pli=1`
[35] Official Jena documentation: `http://jena.sourceforge.net/documentation.html`
[36] RDF On the Go project: `http://code.google.com/p/rdfonthego/`
[37] BSD 2-Clause License: `http://www.opensource.org/licenses/bsd-license.php`
[38] Oracle Berkeley DB: `http://www.oracle.com/technetwork/database/berkeleydb/overview/index.html`
[39] At the time of writing, the download server `http://rdfonthego.googlecode.com/files/RDFOnTheGo.apk` responded with a `404` code: "The requested URL `/files/RDFOnTheGo.apk` was not found on this server" (Accessed on 17th June 2011).
[40] RDF on the Go Wiki: `http://code.google.com/p/rdfonthego/source/browse/wiki/GettingStarted.wiki`
[41] See `http://code.google.com/p/rdfonthego/downloads/list`
[42] SWIP: semantic web in the pocket: `http://swip.inrialpes.fr/`
[43] ContentProviders: `http://developer.android.com/guide/topics/providers/content-providers.html`

storage and is currently in prototypical status [DE10]. For demonstration purposes, data stored in device-internal data sources such as calendar entries or contacts have been exposed as RDF-based Linked Data and visualized through a generic browser interface. A number of extensions have been developed for the SWIP architecture (cf. [DE10]): an *RDFContentProvider* exposes device-internal data as RDF, an *RDFContentResolver* redirects query requests to the respective components, the *Pikoid application* allows for annotating pictures taken by the user with RDF-based metadata, an *AndroidRDFProvider* exposes locally stored data as RDF-based linked data, and an *RDFBrowser* allows for visualizing and navigating RDF data. SWIP is released as a Google Android application (`.apk`-file) and requires a manual installation on the device. Unfortunately, the project's source code was not available for download at the time of writing[44]. Due to this fact, additional details regarding the internal data models, supported Semantic Web languages, inferencing techniques, and serialization formats could not be given. However, information regarding the different components of the SWIP project are available online on the official project page and a description of the architecture together with prototypical details have been published in a conference paper [DE10]. No licensing information could be found on the official web page.

3.3.4 Discussion and Summary

At the time of writing, three RDF frameworks, one RDF/XML parser that has been extended with RDF support, and one XML parser exist for the processing of RDF on mobile platforms. All of the surveyed RDF/XML parsers and RDF frameworks are available as open source software and can be used in both commercial and non-commercial as well as scientific projects; in addition, it is possible to modify and adapt them according to individual requirements. Source code was not available for neither of the two RDF infrastructure frameworks where licensing information could only be obtained for the RDF On the Go project. All RDF/XML parsers and RDF frameworks are released as pre-compiled Java libraries and can be integrated into existing projects without much manual effort. Since the two infrastructure frameworks RDF On the Go and SWIP were developed for the Android platform, they are released as Android applications (`.apk`-files). Quality of APIs varies among frameworks; kXML and NanoXML for J2ME exhibit stable APIs whereas Mobile RDF lacks substantial RDF graph modification operations. Although μJena and Androjena exhibit a relative complete and comprehensive API for processing RDF, RDFS, and OWL ontologies, their APIs are currently in prototypical status with few deficiencies here and there. The same applies to RDF On the Go and SWIP, which—due to their early stages—currently attire the status of research prototypes. All solutions use different internal data structures for in-memory storage and management of RDF data, ranging from `Vector`-based and `HashMap`-based data structures to the implementation of a proprietary in-memory triple store (`FasterTripleStore`) exhibited by the Androjena framework. Further details about the internal data structures, supported Semantic Web languages, inferencing mechanisms, as well as supported serialization formats of RDF On the Go and SWIP could not be provided since their source code has not been released yet. All analyzed frameworks can be deployed on the Android platform; however, NanoXML for J2ME, Mobile RDF, and Androjena can not be used on devices that contain a MIDP or CLDC-based operating environment due to the incompatibilities between the different Java Virtual Machines. Since the technical capabilities of mobile devices are continuously enhancing, we expect more powerful virtual machines to be deployed on next generation devices. Evidently, since RDF On the Go and SWIP were developed for the Android platform, they can only be deployed on devices that feature a Dalvik-compatible

[44] The download server answered with "The requested URL `/code/RDFContentResolver-1.0.zip` was not found on this server." (Accessed 17th June 2011).

virtual machine. kXML is the one and only analyzed parser that does not incorporate RDF processing capabilities; all other frameworks offer native support for RDF where μJena and Androjena are capable of processing RDF/S and OWL ontologies and offer dedicated RDF/S and OWL class libraries. Support for inferencing can not be found in XML parsers; however, all mobile RDF frameworks incorporate inferencing support, differing in its complexity and comprehension, ranging from simple computations of transitive closures on sub-property and sub-class hierarchies to rule-based reasoning where Androjena exhibits the most comprehensive set of inferencing mechanisms. In contrast to RDF On the Go and SWIP, none of the analyzed works offers explicit query support or supports any of the query languages defined for RDF data. In addition, a dedicated mobile storage infrastructure based on a mobile DBMS or a triple store optimized for mobile platforms has not been integrated into any of the surveyed works for the permanent storage of RDF data; instead, only a file-based persistence is supported. On the contrary, RDF On the Go uses an adapted version of the Berkeley database for RDF storage whereas SWIP utilizes the SQLite database of the Android operating system and offers specific interfaces and implementations for RDF data storage in its relational schema. Apart from Androjena, which supports the most popular and established serialization formats, all other frameworks are specialized towards one specific serialization format and require a transformation of the RDF document into this specific format beforehand. Therefore, those frameworks need to make use of converter services for RDF data such as the *rdf:about RDF Validator and Converter*[45] or the *mindswap RDF converter*[46]. All projects are well documented, where some frameworks describe their classes and methods using a Javadoc (kXML, Mobile RDF, μJena) and complement it with external articles or wiki-based systems. Since μJena and Androjena are Jena derivates, the official Jena documentation also applies for them. Extensions are available for only two RDF frameworks, Mobile RDF and Androjena that add query and persistence functionality as well as RDF graph-rendering features. All evaluation results are summarized in Figure 3.1.

Recapitulating, kXML does not provide any RDF support at all and NanoXML for J2ME as well as Mobile RDF lack sufficient RDF management and processing capabilities such as RDF/S and OWL ontology support and manipulations on RDF graphs. In addition, none of the existing mobile XML and RDF frameworks fully supports queries on RDF data via SPARQL or other query languages, although Androjena provides a prototypical implementation of the Jena ARQ libraries. The other RDF frameworks use proprietary query implementations in the form of selectors or iterators that provide only basic and generic query functionality. Besides the TDBoid project that adds a native triple store implementation to Androjena, a native storage mechanism that translates RDF data into internal storage formats used by mobile devices (e.g., the *SQLite* database provided natively by the Android platform) and vice versa could not be found. However, the analyzed RDF storage and query infrastructures are available as experimental prototypes or concept studies to date and lack specific storage and query optimizations for mobile platforms. Nevertheless, they demonstrate that typical RDF processing and storage tasks can be executed on mobile devices although the efficient execution of complex processing operations (e.g., reasoning) or indexing mechanisms is still subject to further research. In summary, Androjena and μJena revealed to be the most mature frameworks for processing RDF data on mobile platforms where μJena yields severe performance issues when processing larger RDF graphs (see Chapter 6).

[45] RDF Validator and Converter: `http://www.rdfabout.com/demo/validator/`
[46] mindswap RDF converter: `http://www.mindswap.org/2002/rdfconvert/`

	XML Parser		*RDF Frameworks*			*Infrastructure Frameworks*	
	kXML	NanoXML for J2ME	MobileRDF	MicroJena	Androjena	RDF on the go	SWIP
License Model	BSD license	zlib/libpng License	Apache License 2.0	GNU General Public License	Apache License 2.0	BSD 2-Clause License	N/A
Deployment and Installation	Library	Library	Library	Library	Library + additional libraries	Android Application (.apk)	Android Application (.apk)
Quality of APIs	Stable release	Stable release	No graph modifications	Prototypical status	N/A	Prtotypical status	Prototypical status
Internal Data Models	Vector	Vector, proprietary data structure	HashMap	Vector	Proprietary data structure (FasterTripleStore[2])	N/A	N/A
Platforms	Android, CLDC, MIDP	Android, CLDC, MIDP	Android, CDC	Android, CLDC, MIDP	Android	Android	Android
Semantic Web Languages	not supported	RDF, OWL	RDF, RDFS, OWL (not natively)	RDF, RDFS, OWL	RDF, RDFS, OWL	N/A	N/A
Inferencing	No	No	RDFS, Rule-based Inferencing	basic RDFS inferencing	Yes	N/A	N/A
Query support	No	Internal	Internal	Internal	Internal	SPARQL	SQL
Persistence	No	No	No[1]	No[1]	No[1]	Yes (Berkeley DB)	SQLite Database
Serialization	XML	RDF/XML	RDF/XML	N-Triple	RDF/XML, N-Triple, N3	N/A	N/A
Documentation	Javadoc, external articles	Wiki	Javadoc, Wiki	Javadoc, Master thesis	N/A, Wiki	Wiki, publication	Webpage, publication
Source Code	Yes	Yes	Yes	Yes	Yes	No	No
Extensibility	N/A	N/A	GraphML	N/A	ARQoid, TDBoid	N/A	Pikoid
Latest Release	June 2006	May 2008	September 2008	2008	Dec. 2010	March 2011	N/A
Current Version	v2.3.0	v1.0	v0.3	v1.5	0.5	N/A	N/A

[1] serialization and writing in a file-based output stream
[2] proprietary tripel store implementation

FIGURE 3.1: Summary of the evaluated mobile XML parsers, RDF frameworks, and query and persistence frameworks (summarized as *'Infrastructure Frameworks'*)

3.4 Analysis and Review of Related Projects

In this section, we review and analyze related Semantic Web projects that are designed for mobile information processing and incorporate context-aware functionality. We investigate how semantic technologies and context awareness are synthesized in those systems and also address limitations and peculiarities of related works. In this survey, we distinguish between Semantic Web-based applications that incorporate context-aware features and Semantic Web-based context frameworks. All projects have in common that they incorporate technologies and languages from the Semantic Web for implementing context-aware functionalities. Since the notions of context and context awareness are exhaustively surveyed in domains such as pervasive and ubiquitous computing (cf. Section 2.4), this analysis specifically focus on context-aware applications and projects that emerged in the Semantic Web domain. In the following, we introduce context-aware features that serve as criteria for classifying the considered works according to the context-relevant features they provide.

- *Quality of Context.* In mobile or ubiquitous computing, context is often acquired in a distributed way and from heterogeneous context sources, where a context source might become temporarily unavailable or deliver incomplete or inaccurate data [PvHS07]. To deal with this inherently present variance in representing contextual information, the notion of *Quality of Context (QoC)* is proposed as a metric to determine the trustworthiness and accuracy of acquired context information and to support context-aware applications and frameworks in selecting those context information the best match predefined requirements (e.g. [BS03, BTC06, PvHS07, SWvS07, MTD08]). QoC can therefore be considered as a mechanism to ascertain the quality of contextual information[47]. This feature indicates whether QoC and *context quality indicators* are considered in context processing tasks of the analyzed works to compute the trustworthiness and validity of contextual information for facilitating context aggregation and context reasoning.

- *Query support.* We ascertain whether contextual information is available to external applications (context consumers) and which query languages, query APIs, or protocols are offered in order to utilize the contextual information acquired by a framework or application. We specifically consider query languages that emerged in the Semantic Web domain such as SPARQL [PS08a], RDQL [Sea04], as well as SPARQL-specific extensions[48], that is, aggregation, spatial queries (e.g. GeoSPARQL[49]), and tSPARQL[50] (cf. [Har09]). This also applies to the idea of treating a context framework as a context service (cf. [LSD+02, SFH09]) or a context repository (cf. [HSP+03, Che04]) whose data can be obtained via query interfaces and further processed by a context consumer. In this respect, a context framework or context-aware application can be considered as a context source where query interfaces allows for retrieving processed, aggregated, and consolidated context descriptions. Query support additionally allows clients to query for the information they are interested in instead of processing an entire context description (cf. [EPR08]). So context consumers are able to query for relevant information and omit irrelevant assertions.

- *Dissemination.* In contrast to the feature query support, which analyzes the query languages supported by a context framework or context-aware application, dissemination

[47] Determining the accurateness and validity of contextual information is considered as one of the main challenges of context-aware computing in pervasive and ubiquitous environments [PvHS07]

[48] An overview of SPARQL extensions is given here: `http://www.w3.org/wiki/SPARQL/Extensions`

[49] GeoSPARQL: `http://geosparql.appspot.com/`

[50] tSPARQL: `http://trdf.sourceforge.net/tsparql.shtml`

refers to a system's capability to disseminate contextual information, i.e., to make contextual information available to external applications via dedicated interfaces, APIs, network protocols, or communication technologies. In general, dissemination can be defined as the set of mechanisms and technologies used to propagate contextual information encoded in context descriptions among clients [CK02, GASW07]. We analyze existing technologies and approaches used for context dissemination such as *push-based*, *pull-based*, or *consumer-subscriber models* (cf. [KMK+03]) as well as how acquired contextual information can be used and utilized in general, for instance through dedicated programming language-specific APIs etc.

- *Context Types.* A wide range of classification schemes and classification frameworks have been proposed to group and distinguish the different types of context-relevant information such as location, identity, time, subject etc. (see Section 2.4.2 for an overview). We analyze, which types of context are supported by a framework or context-aware application and expound the primary context types supported by a system.

- *Context Management Architecture.* This feature describes the underlying architecture a context-aware application or framework is built upon (see Section 2.4.3.2 for an overview). We distinguish between (i) *single autonomous applications or systems* that can be locally deployed on a mobile device and operate independently of network or server infrastructure, (ii) *client-server-based architectures* where the client is responsible for context acquisition and the provision of end-user interfaces and the server hosts context processing components, (iii) *middleware infrastructures* that incorporate context-aware features for context acquisition, processing, and management that can be utilized by applications, and (iv) *context server architectures* that host the full spectrum of context processing and management functions and operate independently of any clients. A survey of existing architectures is presented in [BDR07] and [SB08b].

- *Acquisition Architecture.* Context acquisition describes the process of gathering and collecting context-relevant data from local or remote sensors, or other potential context sources such as context repositories or Linked Data sources (e.g. [HSM+10]) that have been exploited by a context framework. In general, the architecture of a context-aware system is significantly determined by its acquisition techniques and acquisition architecture [BDR07]. Several architectures for context acquisition have been proposed (cf. [BDR07, CFJ03, CJ04, Kja07, SB08a]) which can be broadly classified into three different groups (cf. [Che04]): (1) *proprietary architectures* that directly access locally deployed sensors where sensor and driver logic is directly implemented in application code limiting context reuse and exchange, (2) *middleware infrastructures* which employ a layered acquisition architecture (e.g. [GSB02, CJ04, BKL+08a, LFWK08]) that encapsulate sensor details in dedicated components and expose uniform interfaces for context utilization, and (3) *context server architectures* (e.g. [HSP+03]) that operate similar to database management systems and offer remote access to contextual information hosted within a context repository. A detailed discussion about the advantages and limitations of each architectural style can be found in [Che04] and [BDR07]. We ascertain the architectural approach chosen by a framework or context-aware application for context acquisition and classify each system according to the discussed acquisition architecture classification types.

- *Context Aggregation and Reasoning.* Context aggregation denotes the task of collecting and merging context-relevant data encoded in context fragments that are logically or semantically related and have been acquired independently from each other, usually in a distributed way. The rationale of context aggregation and reasoning is to provide an

augmented, consolidated, and elaborated, i.e., high-level representation of independently and autonomously acquired contexts that allow for deducing meaningful and high-level assertions (cf. [HNBr97, LMWK05, LFWK08, Geh08, CCMS10]). Especially when contextual data are acquired from different distributed and heterogeneous sources, context aggregation is an important task in creating a unified and consolidated representation of contextual information. Aggregation is often performed on the basis of rule-based reasoning (cf. [CCMS10]). Therefore, we analyze whether context aggregation is supported by a system and the type of reasoning (e.g. rule-based, case-based reasoning, forward and backward chaining etc.) that is applied for context augmentation and aggregation or whether reasoning is supported at all. In this respect, we also ascertain whether a framework or context-aware application allows to define individual aggregation heuristics or reasoning rules.

- *Context Reuse and Refinement.* The openness, flexibility, and uniform representation of contextual information using Semantic Web languages allows for exchanging, combining, and aggregating contextual information in order to refine and consolidate contextual data. We investigate whether the considered frameworks and context-aware applications support the exchange and reuse of context models or contextual information fragments in order to augment and refine acquired context information. Since context-relevant data acquired from sensors is inherently uncertain and "difficult to interpret by high-level components" (cf. [SP04]), several works proposed approaches for *context refinement* (cf. [SB08a]) such as *sensor fusion* [BC04] or *context clustering* [GSB02] to facilitate context reuse and eliminate uncertainties inherently present in acquired context information.

- *Dynamic Context Discovery and Integration.* The dynamic integration of context sources during run-time is a fundamental requirement of context-aware systems that were developed for pervasive and ubiquitous environments. This requirement is also denoted as *dynamic context-service discovery* [WX06]. Context discovery in general describes the capability to dynamically integrate local or remote context sources such as sensors into a context framework or context-aware application during run-time to exploit the information provided. This requirement is of significant importance when a framework accounts for the non-obtrusive utilization and seamless integration of context sensors whenever they are discovered. Unlike the discovery of context sources, integrating them for exploitation involves techniques such as *query translation* and *query mediation* (cf. [Euz05]), as well as a common representation scheme (cf. [EPR08]). Therefore, we ascertain whether a framework or context-aware application supports the dynamic discovery and integration of new context sources, either manually or automatically to extend its contextual information space. However, this requirement is also denoted as *openness* of a system [EPR08].

- *Context Representation and Evolution.* Several models and representation schemes (see Section 2.4.1 for a discussion) have been proposed for modeling contextual information where the Semantic Web languages RDF/S and OWL have proved to be suitable frameworks for modeling contextual information with the necessary expressivity to represent its interdependency and interrelations. In addition to those languages that were designed for distributed and heterogeneous environments, a number of other models have been proposed for context representation such as key-value models, markup-schemes, graph-based models, logic-based models etc. (cf. [SP04, BDR07, BCQ$^+$07]). For the evaluation of related projects, we specifically consider the Semantic Web languages RDF/S and the various OWL dialects and also ascertain whether existing vocabularies were used for describing contextual constellations or whether specific context vocabularies have been defined. We

also elaborate on questions concerning the schemas used for context representation and whether schema supports *context model evolution* (cf. [TS09]).

- *Persistence, Replication, and Caching.* Depending on the underlying acquisition architecture, contextual information is stored in different locations; for instance, if a system resembles the context server architecture, it must include a context repository to make acquired contextual information available to clients. Instead, if a context framework operates autonomously of any server infrastructure, a mobile database management system is required to host contextual information directly on a device. We analyze whether contextual information is hosted locally on a device, whether it is stored externally in a dedicated repository, or whether a context server is used for context storage. We also ascertain if replication or caching strategies are used to store and provide context-relevant data or data that are transferred to a mobile device based on contextual information. Since a stable network connection or permanent online connectivity can not be presupposed in mobile environments, a system should support local data caching or the replication of relevant data in order to maintain working conditions of a system or mobile application. We therefore analyze whether such concepts are considered in the surveyed systems.

After having discussed the main features against which our evaluation is carried out, we introduce and describe the considered relevant works in more detail in the following sections. Since the synthesis of context-aware computing with concepts and elements from the Semantic Web in the domain of mobile Semantic Web-based information systems is a rather young and rarely exploited research area, not many works exist to date. However, we expect more systems to be developed in the near future as indicated by the growing number of research activities in related domains accompanied by the technical advancements of current mobile devices.

3.4.1 Related Projects and Applications

3.4.1.1 DBpedia Mobile

DBpedia Mobile[51] is a location-centric mobile client application that visualizes data sets from the *DBpedia project* [ABK+07, LBK+09] in a Fresnel-based [PBKL06] Linked Data browser based on the user's current position [BB08, BB09b]. DBpedia is a community-driven effort for extracting structured information from the *Wikipedia project*[52] and exposing this information as RDF data on the Web under the GNU Free Documentation License[53]. DBpedia's main objective is to work towards one of the key challenges of computer science and information integration in particular, i.e., *"stitching together the world's structured information and knowledge to answer semantically rich queries"* on these data [ABK+07]. This allows for asking sophisticated queries such as *"retrieve all soccer players with number 10 on their shirts playing for a club with a stadium larger than 50.000 seats located in a country with more than 50 million inhabitants"* against Wikipedia data (cf. [ABK+07, LBK+09]). Data sets from the DBpedia project are exposed

[51] DBpedia Mobile: `http://wiki.dbpedia.org/DBpediaMobile`

[52] Knowledge bases are playing an increasingly important role in enhancing the intelligence of Web and enterprise search and in supporting information integration. However, the problem of traditional knowledge bases is that they are created for specific domains, in most cases by a relatively small group of domain knowledge experts. Maintaining such information sources is a costly and resource consuming task in particular when the knowledge of a domain evolves. Wikipedia had been proposed to address these issues and has grown into one of the largest central knowledge source, which is maintained by thousands of contributors. The DBpedia project leverages this gigantic source of knowledge by transforming data from the wikipedia encyclopedia into structured information using Semantic Web technologies.

[53] The GNU Free Documentation License: `http://www.gnu.org/copyleft/fdl.html`

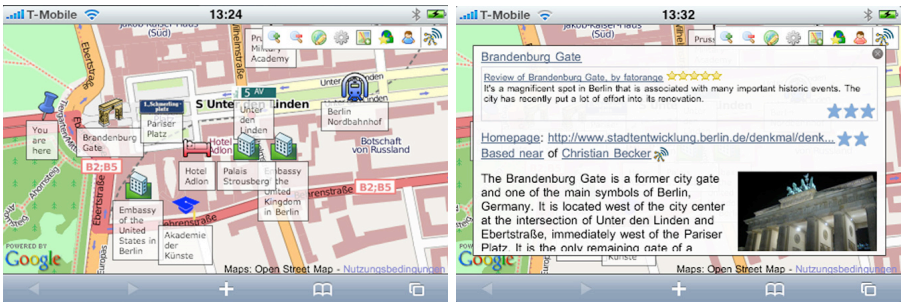


FIGURE 3.2: The DBpedia Mobile client application depicting Semantic Web resources about POIs located around the user's current location on a map (left picture) as well as descriptions of the Brandenburg Gate retrieved from DBpedia and Revyu (right picture)[56]

for both machine and human consumption through a set of APIs and query interfaces. An information extraction framework [ABK+07] converts Wikipedia content to RDF and represents it as a large multi-domain RDF graph, which can be utilized by Semantic Web applications. Furthermore, such data sets can be linked to other data sets exposed on the Web to build a large network of interlinked data sources—the so-called *Web of Data* [BHBL09].

As of June 2011, the DBpedia knowledge base provides information about more than 3.5 million *things* (i.e., resources), including 364.000 persons, 462.000 places, 99.000 music albums, 54.000 films, 17.000 video games, 148.000 organizations, 169.000 species and 5.200 diseases. 274 million pieces of information (RDF triples) are currently stored in the DBpedia knowledge base with labels and short abstracts available in 30 different languages together with 609.000 links to images and 3.150.000 links to external web pages as well as 4.878.100 external links into other RDF data sets. DBpedia further features 415.000 Wikipedia categories and 75.000 YAGO categories[54].

The DBpedia Mobile client takes the GPS coordinates retrieved from the device's GPS sensor or collects information about nearby WiFi networks to calculate approximations of the user's current position and requests information about objects (POIs) located in the user's immediate vicinity from the DBpedia project. Those displayed data sets serve as a starting point for exploring related data that are interlinked with the displayed resources. In this respect, DBpedia Mobile serves as a starting point for exploring the *Geospatial Semantic Web* [Ege02]. DBpedia Mobile maps DBpedia resources to YAGO categories [SKW07]. When a resource is selected, the DBpedia Mobile server retrieves, aggregates, and caches related information from interlinked data sources (e.g. reviews about the selected resource extracted from the Revyu [HM08] project[55]) before they are sent to the mobile device. Additionally, all properties of the selected resource are displayed in a detailed view including incorporated information from other Linked Data sources [BB08]. In case the selected resource exposes RDF links to other resource (cf. [VBGK09]), the user is able to navigate to and browse related Linked Data sources [BB09b]. Figure 3.2 depicts the DBpedia Mobile client application.

DBpedia Mobile is realized as a Java Script application and requires a Document Object Model (DOM) Level 1 and 2 capable browser to make use of the *Google Maps API*[57]. RDF data is not processed directly on a device; instead, data such as the currently visible view area and filter settings are sent to the DBpedia Mobile server that features the *Marbles engine* [BB09a] and uses the

[54]These statistics were retrieved from the official DBpedia homepage (`http://dbpedia.org/About`) on 26th June 2011.
[55]Revyu project: `http://revyu.com/`
[56]Compiled from [BB08]
[57]Google Maps API: `http://code.google.com/apis/maps/index.html`

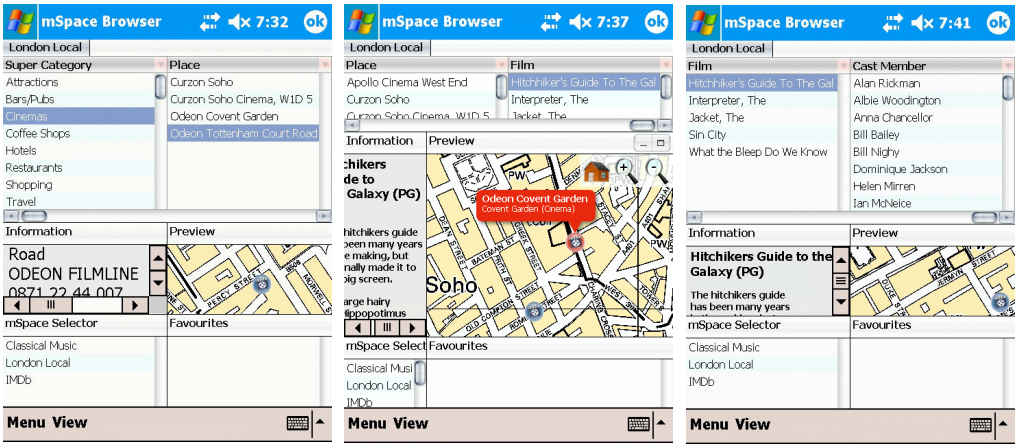


FIGURE 3.3: The mSpace Mobile client application showing cinemas located in the user's immediate vicinity (left picture) together with movie and actor information (center and right picture)[61]

Sesame RDF framework [BKvH02] to transform those data into SPARQL queries [PS08a]. Displayed information as well as related resources are retrieved by dereferencing resource URIs [BB09b] and hosted within an instance of the Virtuoso triple store [EM07] to which such SPARQL queries are issued. To retrieve additional information about interlinked resources, DBpedia Mobile makes use of the *Sindice*[58] [ODC+08] and *Falcons* [CQ09] Semantic Web search engines. All data are retrieved and processed server-side, where the result sets are rendered on the mobile device using a Fresnel-based [PBKL06] Linked Data browser.

DBpedia Mobile also supports simple and SPARQL-based information filtering as well as publishing location-related information such as pictures or reviews to the Web of Data, which is then interlinked with related DBpedia resources [BB09b]. Therefore, each user that registers for the DBpedia Mobile client is provided with a *personal resource URI* that is used for all content contributions of the respective user. Before a view is generated, the server dereferences interlinked resource URIs and retrieves additional data from the Web of Data using the previously mentioned Sindice and Falcons Semantic Web search engines, performs some form of *data augmentation*, and stores the aggregated data as *Named Graphs* [CBHS05] in the Virtuoso triple store [BB09b].

3.4.1.2 mSpace Mobile

mSpace Mobile[59] is a mobile Semantic Web application developed within the *mSpace-project*[60] that uses a multi-faceted column-based browser for exploring data sets that have a direct or indirect relation to the user's current location. mSpace Mobile offers related information about the user's physical environment such as nearby points of interests, amenities, etc. Access to location-based information is offered through a multi-faceted browsing interface that has been specifically adapted to mobile devices. Considered contexts are *time*, *space*, and *subject*. For instance, users are able to receive additional information about the movies currently playing at nearby cinemas. Figure 3.3 displays the mSpace Mobile client application.

[58] Sindice: `http://sindice.com`
[59] mSpace Mobile: `http://research.mspace.fm/projects/mobile/`
[60] mSpace: `http://research.mspace.fm/mspace`
[61] Compiled from [WRS+05]

mSpace Mobile is built upon the *mSpace Software Framework*[62] [sSO+05] designed for the management and exploitation of distributed semantically related resources. The basic design principle behind the mSpace framework is to offer users a software tool that allows for exploring a multi-dimensional information space in multiple ways by leveraging protocols and languages from the Semantic Web accounting for its scalability as a distributed data source. Users are able to navigate along associated items in pre-defined contextual dimensions that have a particular relationship to a selected subject.

In contrast to traditional location-based information systems that use single and proprietary data sources, mSpace Mobile exploits data from freely available semantic data sources that publish information by using RDF such as the Open Guide to London[63] [WRS+05]. mSpace Mobile further supports *context transitions*, i.e., shifting the focus between contextual entities where data retrieval tasks are dynamically adjusted according to the selected (context-relevant) information item. This is a significant difference since most comparable applications merely consider location as the main context. In this respect, the information item the user is interested in becomes the new context. Although the context is not bound to a fixed entity or subject, the underlying context model reveals to be based on a static schema. Support for combined queries, e.g, by combining two subjects is not implemented; users instead can only query for one single item at a time. As a consequence, advanced context-aware features such as *context fusion* (cf. [GSB02],[BC04]), i.e., aggregation of contextual information fragments or entities, or *context refinement* are not supported by default. The mSpace Mobile Framework rather acts as a multi-faceted mobile data browser that allows for navigating along pre-defined contextual dimensions where the underlying interaction model tries to reduce the cognitive load by maintaining the currently active context.

Although mSpace Mobile is designed for mobile usage, it employs a distributed client-server-based architecture where the server-side abstracts over multiple triple stores and is responsible for dereferencing and integrating resources [WRS+05]. mSpace Mobile employs a three-layered architecture: (i) the *mSpace Application layer* hosts functionalities and components for building mobile client applications on top of the mSpace Framework and for generating the queries issued to the mSpace Query Server. The clients themselves are separated from the query generation and translation steps. (ii) The *mSpace Query Server* offers query services that can be utilized by mobile applications in order to query for context-relevant resources. The Query Server uses SOAP, HTTP, and .NET Web Service technologies for its communication with the client applications as well as with the mSpace Knowledge Server. (iii) The *mSpace Knowledge Server* abstracts over a configurable set of RDF repositories and provides facilities for building links between resources residing in different repositories. For this purpose, the Knowledge Server makes use of the RDQL [Sea04] query language for querying RDF data sources. However, RDQL is officially superseded by SPARQL [PS08a] as the de facto standard query language for RDF data. The default data repository is the *3store*[64] triple store [HG03]. The Knowledge Server further allows for combining inherently isolated data sources. In this respect, we can observe an analogy to the Linked Data approach (cf. [Biz09],[BHBL09]) whose objective is to establish semantically meaningful links between RDF resources by using well-known Semantic Web concepts such as assigning unique URIs to resources in order to make them identifiable and thus dereferenceable [Lew07]. The Knowledge Server thus exposes a WWW approach for data provisioning in that it consists of a variable amount of data providers, each being controlled separately, that scale with the availability of potential sources.

[62] mSpace Framework sources: `mspace.sourceforge.net`
[63] The Open Guide to London (superseded version): `http://london.randomness.org.uk/`
[64] 3store project page: `http://sourceforge.net/projects/threestore/`

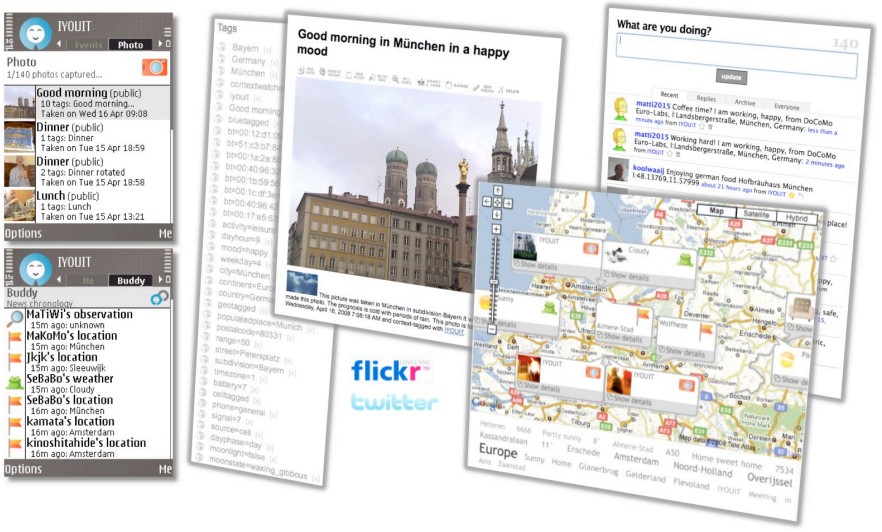


FIGURE 3.4: An overview of the IYOUIT system consisting of two screenshots of the client application (left side) as well as screenshots depicting different views of the community portal (right side)[67]

3.4.1.3 IYOUIT

A rather promising project that aims to combine context-aware functionality and Semantic Web technologies for mobile information systems is *IYOUIT*, which was developed by DoCoMo Euro-Labs[65] Munich in cooperation with the Telematica Instituut[66] Enschede. IYOUIT describes itself as a *"[...] prototype service to pioneer a context-aware mobile digital lifestyle and its reflection on the Web"* [BKL+08a]. It is built on a distributed infrastructure incorporating semantic technologies and languages to allow for a qualitative interpretation and evaluation of user activities, which are acquired locally through the mobile device, processed on a server, and reflected in a community portal on the Web (see Figure 3.4). Those activities are captured by quantitative sensors and mapped to qualitative data abstractions using formal ontologies. Formalized domain knowledge together with classification and ontology-based reasoning mechanisms are used to support the process of deriving meaningful interpretations of gathered raw sensor data as well as the recognition of behavioral patterns (cf. [LMWK05, LFWK08]).

The central idea of IYOUIT is that users can establish relationships among each other and hence build social networks for sharing context data through the IYOUIT Web portal[68]. Those social networks are represented as OWL ontologies so that Description Logic-based reasoning (cf. [BCM+03]) can be applied for detecting inconsistencies and contradictions in social network data to maintain data accuracy, and deducing, i.e., revealing implicit relationships between users in a social network. IYOUIT is designed as a service consisting of a mobile client application and a server infrastructure, the *Context Management Framework* (cf. [BKL+08a]). The mobile client application was developed for the Nokia Series-60 devices and automatically captures and collects information about the entities surrounding a user—the user's context—in order to share personal experiences. Such information is, for instance, the places one visited, the people met, or personal information such as recently read books. IYOUIT uses locally deployed sensors to collect context-relevant data automatically and sends it to the Context Management Framework

[65] DOCOMO Communications Laboratories Europe GmbH: `http://www.docomoeurolabs.de/`
[66] Telematica Instituut: `http://www.telin.nl/index.cfm?language=en`
[67] Taken from [BKL+08c]
[68] IYOUIT Web portal: `http://www.iyouit.eu/portal/`

server onto which such data are interpreted, aggregated, further processed, and stored. IYOUIT also employs interfaces to Web 2.0 applications such as Flickr[69] and Twitter[70] to collect personal information and sharing it online. The main conceptual components incorporated in the system are described as *application domains* where each domain is responsible for the acquisition and processing of a certain type of context (cf. [BKL+08a]):

- *Share:* community-based context sharing synthesizes context awareness and social networking services in order to analyze personal context histories and to discover relationships for identifying potential social networking activities by using formal ontologies and ontology-based reasoning for their representation (cf. [WZGP04]).
- *Life:* the life domain is targeted towards analyzing user-generated content (e.g, tags being manually attached to photographs) and their metamorphosis in the temporal dimension by information extraction and information clustering techniques in order to enable user guidance and aggregate location-based information.
- *Blog:* the blog dimension analyzes blogging capabilities in order to derive and manage complex contextual events using classification-based reasoning [GS91] to capture and interpret the users' current situations (e.g., to describe their current locations such as *in office* or *at home* using high-level ontological concepts).
- *Play:* the play dimension is concerned with identifying and describing contextual constellations by observing context histories and to transform and manage low-level quantitative data in order to *"discover homogeneous time segments in a higher-dimensional context space and to detect correlations between different context dimensions"* [BKL+08a], p.806.

The context management framework represents a layered architecture and network of distributed and interconnected components for collecting, managing, and distributing context information proactively [BKL08b]. It allows for the implementation of flexible services that track, for instance, the position of "friend", identify frequently visited places, collect information about publicly available WLAN hotspots, local weather information, photos, and reflects this information (so-called *context streams*) on the Web portal as well as on mobile clients [BKL+08a, BKL08b, BKL+08c]. Its objective is to transform quantitative raw context data (e.g., sensor outputs) into qualitative, human-interpretable statements reflecting the user's current situation by context aggregation, combination, and reasoning [BKL08b].

The framework includes a *privacy manager* for controlling the distribution of sensitive personal information, an *identity manager* for the connection to and authentication by 3rd party applications such as Flickr or Twitter, a *relation manager* being responsible to reason about the social networks of users, and an *ontology manager* for the utilization of domain-specific knowledge being formalized in core ontologies. It further employs the concept of *context providers* that represent wrapping components for specific context sources and contain aggregation heuristics in order to abstract over low-level quantitative data. An overview of the constituting components is given in [BKL+08a, BKL08b].

A set of core context ontologies based on the Web Ontology Language (OWL) [OWL04] has been developed for context clustering and context data transformations, which are utilized by context providers. Those context ontologies are exclusively used for high-level context elements due to the fact that ontologies are not well suited for handling large amounts of data [WLL+07]. Reasoning

[69] Photo community portal Flickr: `http://www.flickr.com`
[70] Twitter homepage: `http://twitter.com`

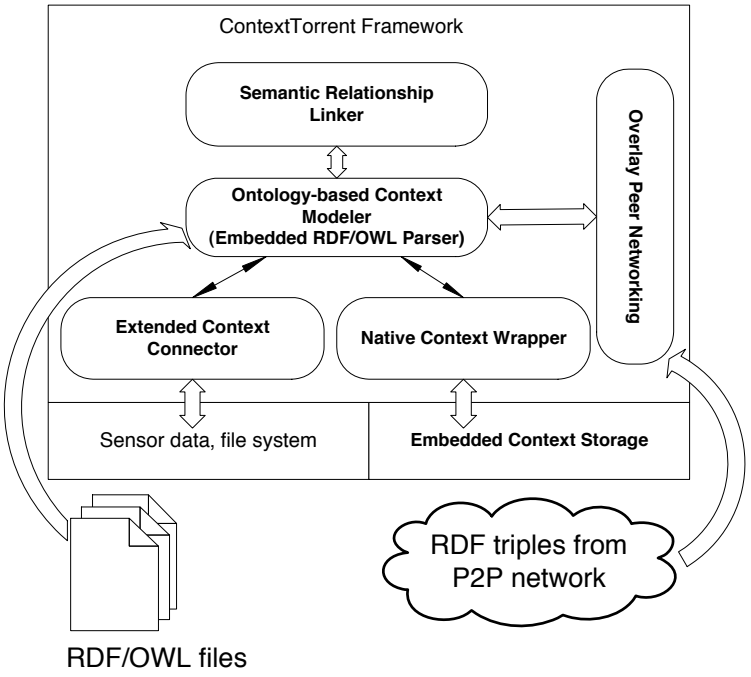


FIGURE 3.5: The ContextTorrent system architecture and its core components[72]

is performed to make implicit knowledge explicit where each relationship has an additional attribute that indicates whether a relation has been explicitly asserted or inferred [BKL08b].

3.4.1.4 ContextTorrent

A very similar system to the one presented in this thesis has been developed within the *ContextTorrent*-project [HDW09]. ContextTorrent is a semantic context management framework that offers access to semantically represented context information for local and remote context-aware applications. It is developed for the Google Android platform and makes use of the mobile Java-based XML parser *NanoXML*[71]. ContextTorrent was inspired by semantic desktop research in which concepts and technologies from the Semantic Web are used to enhance personal information management (cf. [BS04]) by providing context-relevant information in an automatic and proactive fashion to support the user's long-term memory (cf. [SBD05],[FAS09]).

ContextTorrent offers a controlled interface for the exploitation and utilization of semantically represented context data where a mobile device takes the role of a context provider as well as a context consumer. An *overlay peer-to-peer network* allows for connecting mobile devices and mobile applications for large-scaled local or remote context query and provision [HDW09]. The underlying infrastructure allows for building dynamically established semantic links between related context fragments. An *ontology-based semantic modeler* represents contextual data as RDF resources using an adapted OWL/RDF parser and maintains the links between semantically related context fragments in a dynamic fashion. Those semantic relationships are stored in an *object-oriented database* specifically designed for resource-constrained mobile devices which

[71] cf. Section 3.3.1
[72] Taken and adapted from [HDW09], page 333.

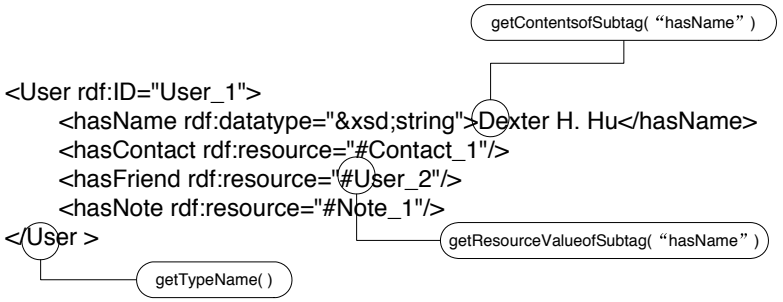


FIGURE 3.6: An example of NanoXML's functions for accessing OWL and RDF document elements[74]

exposes minimal overheads and better resembles ontological representations and corresponding schema evolutions [HDW09]. Figure 3.5 provides an overview of the main architectural components of the ContextTorrent framework.

As opposed to other context management frameworks which distinguish between low-level and high-level or inferred versus aggregated contexts, the management infrastructure employed by ContextTorrent treats all contexts equally regardless of their type or origin where context-relevant information is considered a semantic resource irrespectively whether the information stems from an external source, a sensor, or the internal context repository. An *N-gram based matching algorithm* [MSLN00] is used to rank context information according to its relevance to issued context queries.

The core component of the ContextTorrent framework is the *Ontology-based Context Modeler* which is responsible for modeling and representing contextual data as RDF resources by making use of the included OWL/RDF parser. Such contextual data may either be retrieved remotely via the *Overlay Peer Networking* component that allows for the large-scaled exchange of context information between communicating peers, or acquired from the *Semantic Relationship Linker* which establishes and manages the semantic relationships that possibly exist between context fragments. The *Extended Context Connector* and *Native Context Wrapper* components implement unified interfaces for accessing and wrapping context sources that are either deployed on the mobile system in the form of local file system elements or locally deployed sensors (Extended Context Connector), or application-dependent context information that is stored in the *Embedded Context Storage* component. This component is responsible for storing contextual information including existing semantic relationships in an object-oriented database management system (ODBMS) since it better resembles ontological representations and corresponding schema evolutions.

ContextTorrent uses the Web Ontology Language (OWL) [OWL04] for describing and establishing the relationships among context entities. Context entities are represented as Semantic Web resources that have a unique identifier as well as URI assigned to it to make them distinguishable and referable (cf. [SC08]). ContextTorrent further allows for the dynamic binding of shared context data to applications by making use of the Android's concepts of Intents and Intent Filters[73] in order to enable context sharing between applications at runtime. ContextTorrent distinguishes between static and dynamic contexts where the classification is not based on the type of context data but rather on the frequency of context value changes.

[73]Intents and Intent Filters: `http://developer.android.com/guide/topics/intents/intents-filters.html`
[74]Taken and adapted from [HDW09], page 335.

For processing RDF and OWL data, the open-source Java-based XML parser NanoXML (cf. Section 3.3.1) has been ported to the Android *Dalvaik Virtual Machine* (DVM). Although NanoXML is a lightweight and fast RDF/XML parser, it lacks sophisticated RDF processing and management capabilities (cf. Figure 3.6) compared to other mobile RDF frameworks such as Androjena, μJena, or MobileRDF (cf. [ZS10]).

3.4.2 Analysis

In the following, we discuss the results which have been acquired in the course of our analysis; all results have also been summarized in Table 3.7.

Quality of Context Explicit information whether DBpedia Mobile, mSpace Mobile, IYOUIT, and ContextTorrent use quality of context indicators for context retrieval and aggregation could not be found. Therefore, we assume that quality of context indicators as proposed by, e.g., [BS03, BTC06, PvHS07, SWvS07, MTD08] are currently not implemented in each of the analyzed projects or are used internally and implicitly in server-side components. However, ContextTorrent applies relevancy ranking algorithms to rank requested resources according to their relevance for a given query where the relevancy ranking can be considered a special form of context quality indicator for selecting the resources that match a given request with the greatest possible accuracy.

Query support DBpedia Mobile supports a simplified set of SPARQL queries as well as the manual specification of filtering rules based on YAGO categories [SKW07]. Users are able to issue SPARQL queries manually via a client query user interface to the DBpedia Mobile server, whereas a fully functional SPARQL endpoint (per definition) that can be utilized by external applications is not available, which impedes context exploitation. mSpace Mobile supports the RDF Data Query Language (RDQL) [Sea04] and uses query translations and transformations to communicate with the mSpace Query Server. However, further details about the query transformations are not given. Due to the project's release time, other query languages such as SPARQL as the de-facto standard query language are not supported. IYOUIT allows to query for historically aggregated context data although no information is given regarding supported query languages or query endpoints. ContextTorrent supports the RDF Query Language (RQL) [KAC+02] and other query languages and formats, which have not been specified more precisely in respective works. For the local exploitation and utilization of contextual data, ContextTorrent offers an SQL-based query interface since the Android-specific concept of Content Providers[75] is used for local context queries; Content Providers expose an SQL-based query interface that returns data in the form of relational database tables.

Dissemination Since DBpedia Mobile and mSpace mobile are designed as self-contained location- and context-aware Semantic Web applications, they do not offer any external dissemination features such as APIs or communication interfaces for the contextual information acquired or generated by these applications; context information is exclusively disseminated internally between the mobile client and the corresponding server infrastructure. However, the DBpedia Mobile server can also be accessed online via a web application[76]. mSpace Mobile uses `.Net` Web Service technology and the SOAP protocol [BEK+00] for context dissemination.

[75] Content Providers: `http://developer.android.com/guide/topics/providers/content-providers.html`
[76] DBpedia Mobile as web application: `http://beckr.org/DBpediaMobile/`

	DBPedia Mobile	mSpace Mobile	IYOUIT	ContextTorrent
Type	Application	Application	Application	Framework
Quality of Context (QoS)	Not supported	Not supported	No information available	Implicitly supported through relevance ranking algorithms
Query Support	SPARQL and individual filtering rules	RDQL, specific Query APIs	No information available	RQL and other query languages that have not been specified in more detail
Dissemtination	No external APIs available; the DBpedia Mobile server is accessible via a Web application	The mSpace Knowledge Server offers .NET Web Services, SOAP, and HTTP interfaces	No supported; contextual information is only transferred internally between the mobile client and the IYOUIT server	Externally via the peer-to-peer networking layer and internally through Android-specific concepts of Intents and Intent Filters
Context Types	Location and user-defined resources	Location and predefined contextual dimensions related to location-based information	Multiple context types defined in the IYOUIT context ontologies	No preference for a specific context type; all types are treated equally
Application Architecture	Client-Server Architecture	Client-Server Architecture	Client-Server Architecture	Middleware Architecture
Acquisition Architecture	Proprietary	Proprietary	Middleware-based architecture; acquisition through context providers or manual input	Middleware-based acquisition architecture (sensor and peer-to-peer network layer)
Context Aggregation and Reasoning	Server-side aggregation and reasoning; no support for individual reasoning rules or aggregation heuristics	No information available	Server-side aggregation using multiple OWL DL reasoners and reasoning techniques	Context aggregation and reasoning is not implemented in the current release
Context Reuse and Refinement	Not supported	Not supported	Supported only server-side	No explixit support for reuse of contextual information
Dynamic Context Discovery and Integration	Not supported	Not supported (dynamic integration requires an adaptation of the mSpace framework and the mSpace Knowledge Server)	No information available whether an integration of new context providers requires an adaptation of the underlying infrastructure	Framework instances can serve as a context sources and are integrated via the peer-to-peer network layer
Context Representation	Context is represented as XHTML client-side and as RDF/S and OWL on the server-side	Context is represented as RDF although no information regarding the used vocabularies for representing contextual information is given	Low-level contexts are represented in proprieaty formats; high-level contextual information is represented server-side using RDF/S and OW	Contextual information in general is represented as resources using RDF/S and OWL
Persistence	Server-side using a Virtuoso triple store	Several data stores hosted within the mSpace framework server-side	Server-side in the IYOUIT server	Local file system of a mobile device or in an object-oriented database
Replication	Not supported	Not supported	Not supported	Not supported
Caching	Server-side	Not supported	No information available	Not supported

FIGURE 3.7: Summary of the evaluated Semantic Web projects

However, contextual information can only be requested from the mSpace Knowledge Server or the mSpace Query Server rather than from the mobile client directly. In the IYOUIT project, contextual information is transferred only internally from the IYOUIT server to the mobile client and vice versa. We could not find any information whether contextual information acquired by IYOUIT clients can be exploited by 3rd-party applications. ContextTorrent disseminates contextual information externally via the peer-to-peer networking layer or internally by making use of the Android-internal concepts of Intents and Intent Filters (cf. [Mei10]).

Context Types DBpedia Mobile and mSpace Mobile use location as the main contextual information type; users can navigate through resources located in their immediate vicinity where the selected resource becomes the new context for retrieving additional information. The types of context processed by the IYOUIT context management framework are on one hand determined by the context providers deployed within the framework but also through the context ontologies defined in the course of the project (see [BKL+08a]). However, those ontologies are not available publicly wherefore no further information regarding the supported context types could be obtained. For efficiency and scalability reasons, not all contextual information entities are represented using ontological concepts since ontologies in general are not well suited for handling large data amounts efficiently (see [WLL+07, BKL+08a]). ContextTorrent is a generic context management framework and is intended to support a wide range of contextual information of different type. ContextTorrent treats all types of contextual information equally.

Application Architecture DBpedia Mobile, mSpace Mobile, and IYOUIT are built upon a client-server-based architecture where the client is used for acquisition, interaction, and visualization purposes and the context management and processing logic is executed server-side. ContextTorrent, in contrast, is developed as an autonomous mobile middleware infrastructure system that operates irrespectively of a server infrastructure. All context processing tasks are performed directly on the mobile device.

Acquisition Architecture DBpedia uses a proprietary application for accessing locally deployed location sensors on a device or alternatively tries to infer the user's location via WiFi-based location ascertainment. mSpace Mobile accesses the device's built in GPS sensor to gather information about the current location wherefore it applies to the proprietary architecture classification. IYOUIT also uses a mobile client application for context acquisition that encapsulates sensor driver logics for raw-data retrieval. It is built on a flexible acquisition architecture by requesting context-relevant data either implicitly from locally deployed sensors or explicitly in case the user manually enters context-relevant information. ContextTorrent employs a middleware-based architecture for context acquisition where contextual information can be acquired from locally deployed sensors, a context repository, or from another mobile device running the ContextTorrent framework via the overlay peer-to-peer network layer.

Context Aggregation and Reasoning In DBpedia Mobile, the aggregation of context-relevant information is performed server-side using RDF/S and OWL-based reasoning. No information whether mSpace Mobile supports context aggregation or context reasoning could be found. IYOUIT uses aggregation and reasoning techniques for transforming low-level quantitative data into high-level qualitative contextual information where the task of aligning raw context data to qualitative contextual concepts is solely left to each individual context provider. For deriving higher level data abstractions, each context provider can implement an interface

to an OWL reasoning component that allows for qualitative value classifications using standard Description Logic techniques [BKL+08a]. The aggregation of context-relevant information is performed server-side using RDF/S and OWL-based reasoning. According to the literature available about the ContextTorrent system, it does currently not provide any form of context aggregation or context reasoning. Reliable information whether one of the analyzed context-aware applications or frameworks provides support for the definition of individual reasoning rules or aggregation heuristics could not be found.

Context Reuse and Refinement Since DBpedia Mobile and mSpace Mobile are per definition context-aware Semantic Web applications that offer location-based information retrieval services, context reuse and refinement as such are not considered primary functions and hence are not supported actively. However, since both projects allow to navigate along contextual dimensions where the selected resource is considered as the currently active context, transitions between context-relevant resources can be considered as context refinements in a similar sense. IYOUIT mutually shares contextual information between components of the context management framework for reuse and refinement purposes. The ContextTorrent framework does not support the reuse of contextual information for context refinement at the moment but allows for establishing semantic relationships between contextual fragments during runtime.

Dynamic Context Discovery and Integration DBpedia Mobile is built as a self-contained application where location is the primary contextual information and hence does not support dynamic context source discovery and integration. Additionally, mSpace Mobile and the mSpace Framework do not support the dynamic integration of new data sources on the fly since the integration of new context or data sources requires an adaptation of the mSpace framework. This renders the on-the-fly integration impossible without explicit human involvement. For IYOUIT, explicit information whether new context providers can be dynamically integrated during run-time could not be found. Since an integration of new contextual data sources requires an adaptation of the client application as the primary context collector, we assume that a dynamic integration is currently not supported. The same fact also applies to the ContextTorrent framework where also no explicit information regarding the dynamic discovery and integration of context sources could be found. From the fact that the definition and integration of new context types requires "little implementation work" (cf. [HDW09]) we can deduce that the dynamic integration of new context sources during runtime is currently not supported. However, the acquisition based on the peer-to-peer networking layer can be regarded as a form of dynamic context discovery, where another mobile device servers as context source or context repository.

Context Representation and Evolution Contextual information as such is not represented explicitly using Semantic Web languages in DBpedia Mobile; although when the user starts to browse for interlinked resources, that information is retrieved from Linked Data sources and stored as RDF data server-side, but transformed into the XHTML format by the Marbles engine [BB09a] before it is sent to a mobile device. mSpace Mobile represents contextual information server-side as RDF data although no information regarding the used vocabularies for expressing context-relevant data is given or whether these data are transformed to a specific format to be processed and displayed on the client. IYOUIT represents high-level contextual information using RDF/S and OWL based on a set of defined context ontologies (see [BKL+08a, BKL08b]); low-level or raw-sensorial data is represented both in proprietary data formats as well as using elements from Semantic Web languages and vocabularies. However, context providers are able

to make use of the internal DL reasoner for context aggregation tasks. Within the ContextTorrent framework, contextual information is represented using RDF/S and OWL but no further information w.r.t. the used vocabularies is given. Additionally, none of the analyzed systems provides explicit information regarding context model evolution.

Persistence, Replication, and Caching DBpedia Mobile uses an instance of the Virtuoso RDF store to store and aggregate context-relevant data, which is retrieved from related linked data sources. The Virtuoso server also supports a form of pre-caching so that queries are first answered by the local database before other data sources are queried. To the best of our knowledge, DBpedia Mobile does not cache or store data locally on a mobile device. In mSpace Mobile, contextual data is stored in multiple data sources (triple stores) using the mSpace framework. In this respect, a triple store acts as a *data provider* and is requested independently using a WWW approach [WRS+05]. No information regarding preemptive caching of contextual information could be found. However, to the best of our knowledge, no information is cached locally on a device. IYOUIT stores and hosts context-relevant information server-side in the Context Management Framework and does not replicate context-relevant data to the mobile device; instead, context data updates are reflected in the user interface of the IYOUIT client application. The ContextTorrent framework allows for storing contextual information on the local file system or within a locally deployed object-oriented database[77]. To the best of our knowledge, it does not implement any form of replication or caching strategies.

3.4.3 Summary

Most approaches that aim at using Semantic Web technologies for the implementation of context-aware functionality exist as self-contained systems that exhibit limited sharing capabilities. Contextual information is predominantly hosted in external triple stores where a client application acts as acquisition and visualization interface. Although almost all analyzed works use Semantic Web-based description frameworks and languages for context representation, they exhibit only a limited set of client or query interfaces that allow for using acquired contextual information externally or to make use of context-aware functionality. Context aggregation and reasoning steps are performed server-side. As contextual information is processed only internally, explicit information about quality of context aspects could not be found. Moreover, most context-aware Semantic Web applications implement context-aware functionality for a specific purpose where a framework that specifically focuses on mobile RDF data replication could not be found.

3.5 Conclusion

In summary, our analysis revealed that context-driven replication of RDF data to mobile devices has not been addressed by current or related research yet. Existing replication algorithms are considered complementary to our approach as they focus on specific context parameters rather than on the entire user context. Replication architectures that follow a more generic strategy require the employment of server infrastructures for context processing and the execution of reasoning heuristics. The RDF parsers and frameworks currently available for mobile systems mostly exist as research prototypes but provide the necessary functions for a context-dependent RDF data replication infrastructure as proposed in this thesis although much space is left for

[77] For a performance evaluation, the two object-oriented database systems *Perst* (`http://www.mcobject.com/perst/`) and *db4objects* (`http://www.db4o.com/`) were analyzed.

optimization. In Section 6, we therefore analyze the performance of mobile RDF frameworks in typical replication-related tasks comprising the creation, parsing, modification, and storage of RDF data replicas directly on a mobile device. Only two frameworks offer advanced RDF processing and management capabilities such as inferencing but lack a sophisticated query support; however, there exists a prototypical implementation of the SPARQL query standard for the Androjena framework. The mobile storage and query frameworks that exist to data are currently available as experimental research prototypes but recent developments indicate an increasing awareness of deploying Semantic Web technology on mobile devices (cf. exploiting Linked Data for mobile Augmented Reality [RHP+10], SWIP [DE10], or i-MoCo [WBB08]).

The context-aware Semantic Web applications we analyzed extensively use Semantic Web-based description frameworks such as RDF or OWL but outsource processing-intensive tasks to external servers or to specific server-side applications rather than execute them on the device itself. This means, however, that in case of missing network connectivity the client applications become practically useless. While our architectural approach does not allow to proactively update data from remote sources without connectivity, it provides at least a local buffer of the data that have been replicated so far, and hence allows the user to continue using the applications, although in a restricted manner. Another distinct aspect is that context acquisition and context representation is not limited to a predefined set of contextual aspects, i.e., the context descriptions created by the framework are dynamic and include as many aspects as could be acquired. Applications can process the data they are interested in, leading to a greater flexibility in elaborating on contextual constellations.

Chapter 4

Approach

"Progress is the product of human agency. Things get better because we make them better. Things go wrong when we get too comfortable, when we fail to take risks or seize opportunities."

Susan Rice, Stanford University Commencement, 2010

After having discussed recent projects from the Semantic Web domain that aim to synthesize context-aware computing concepts with semantic technologies for building more user-oriented applications and services in the previous chapter, we now introduce a conceptual architecture of a context-sensitive RDF data replication framework for mobile devices on a formal basis. We both formally define and conceptually describe the abstract model and algorithms that constitute our chosen approach and also present requirements as well as design considerations that constitute the architectural fundament of the proposed context-sensitive replication framework. The hypothesis motivating our work is that *a combination of context-aware computing concepts and semantic technologies to built a Semantic Web-based context-sensitive replication framework for RDF data on mobile devices allows for increasing the accurateness of proactive and transparent context-dependent information retrieval processes to better accommodate the information needs of mobile users*[1]. A central aspect of our approach is thus the reliance on technologies, concepts, and languages that emerged in the Semantic Web domain and introducing such concepts to mobile platforms in particular for the processing and replication of RDF data sets to such platforms. Although this work emphasizes the use of semantic technologies for the representation and processing of contextual information as outlined in Chapter 3, the formal and conceptual descriptions of the framework's main constituting components and workflows occur principally independent from such technologies. The main contributions of this work, as outlined in Section 1.4, therefore are the specification of a formal model together with a conceptual system architecture for the distributed acquisition, aggregation, consolidation, local storage, and dissemination of independently acquired heterogeneous contextual information that serve as a basis for the proactive and transparent replication of RDF data sets to mobile devices in order to support the information needs of mobile users. Our approach demonstrates that a synthesis of such concepts and technologies not only allows for the development of new sophisticated mobile

[1] As outlined in Section 1.1.1, 72% of mobile users find their information needs inadequately addressed whereas a share of 58% can be satisfied by retrieving data from publicly available sources (cf. [KB06, SLGH08]). The relevance of context in regard to the information needs of mobile users has also been investigated in related studies such as [CS08].

applications and services that could satisfy the user's situational information needs in a proactive, transparent, and ad-hoc manner but also leads to a new form of context-aware computing that we have denoted as *Semantic Web-based context-aware computing* in [ZS12b].

In the remainder of this chapter, we therefore present a formal model of our approach together with a formal definition of the orchestration logic for the dynamic orchestration of context acquisition components called *context providers* based on a data description ontology. The data description ontology allows to describe the data emitted by a context provider in a unified and well-defined way and serves as a basis for the calculation of similarity scores in order to route data between compatible context providers for context refinement, augmentation, and complementation. We algorithmically describe the process of deducing orchestration networks of compatible context providers and also present a transactional processing model for the distributed and autonomous acquisition and aggregation of context-relevant information while maintaining data and process consistency as well as data accurateness and data completeness under consideration of the peculiarities imposed by mobile operating system infrastructures. For the description of the formal model underlying the conceptual system architecture, we use algebraic specifications and symbols to refer to and denote constituting elements.

However, before we introduce the formal model and conceptual architecture of the proposed framework and formally define its main constituting concepts and building blocks, we present a number of requirements collected and aggregated from the relevant literature that serve as design considerations of our system. We conclude this chapter with a summary and a discussion of selected requirements we consider essential to our work that concern the completeness and consistency of contextual information as well as the integrity of aggregation and processing workflows. The concurrently operating transaction-based processing model formally introduced in the penultimate section guarantees completeness, consistency, and accurateness of contextual information and yields a deterministic processing behavior even in uncontrolled situations[2]. The conceptual specification of the system's architecture in conjunction with the formal model can be used as a basis for an implementation of the proposed framework in a specific programming language and for a specific mobile platform[3].

4.1 Requirements and Design Considerations

The realization of context management and context processing functionality in form of a framework-based approach is broadly suggested in the relevant literature since a framework-based architecture not only facilitates and stimulates context-aware application development, it further enables the gathering, interpretation, and aggregation of contextual data in a structured, well-defined, and controlled way [FMGI06]. The context-sensitive replication framework proposed in this thesis has been developed in the course of the MobiSem project[4] and extends such broadly suggested framework-based approaches in that it has been specifically designed to operate on mobile systems while making use of Semantic Web technologies to acquire, interpret, reuse, aggregate, store, disseminate, and reason on contextual information independent of any application or operating system-specific infrastructure. Semantic Web technologies and practices, which are designed as an information processing infrastructure for heterogeneous environments, can help

[2]Mobile ad-hoc environments or ubiquitous environments are inherently considered as being unpredictable and uncontrolled.

[3]In Chapter 5, we demonstrate the implementation of selected aspects defined in the formal model on the Android platform.

[4]Information about the MobiSem project can be found at the official web site: `http://www.mobisem.org`.

in solving the issues outlined in Section 2.4 and 2.5, and are therefore a crucial constituent for future context-aware systems operating in ubiquitous and mobile environments [ZS12b].

In the course of this chapter, we denote and refer to the system proposed in this thesis as *context framework* or, more formally, as *context-dependent replication framework*. For the design of this system, we have collected and consolidated a number of requirements and fundamental properties of comparable systems being published in the relevant literature (cf. [DAS01, Win01, BC04, HIMB05, EPR08]). These requirements have been adapted for mobile operating systems and mobile platforms in particular and serve as design principles of our work. In the following, we provide a consolidated overview of each requirement and briefly discuss the aspects of a mobile context processing and management framework being addressed by a specific requirement; we then present the high-level goals of our framework and outline its main advantages w.r.t. comparable client-server-based approaches.

- *Flexible system design and open system architecture.* A context management framework for mobile and ubiquitous computing environments must support the integration and exploitation of dynamically discovered heterogeneous context sources at run-time using standard technologies to handle technical, structural, and semantic heterogeneity. This requirement not only accounts for the flexible integration of context sources but also concerns the framework as such and the schemas used to describe and represent contextual information. It is referred to as *openness* in [EPR08]. Moreover, a framework should be capable of adapting its processing tasks and strategies according to operational circumstances and available system resources.

- *Open and flexible context representation models.* The context descriptions produced by a context framework should be based on a dynamic and flexible schema that allows for incorporating contextual data that have been acquired from context sources that were not anticipated at design time of a context framework. This process should be accompanied by languages and vocabularies from the Semantic Web due to their openness, their adherence to the *open world assumption* (cf. [DS06]), as well as their reliance on standard protocols and techniques in order to facilitate the interoperability among context producing and context consuming components. Additionally, it should be possible to describe new types of contextual information with existing and individual semantic vocabularies so that those descriptions can be processed by different context consumers.

- *Minimal platform dependency.* A context management framework for mobile device should be easily deployable on existing mobile systems and infrastructures while taking into account available system resources. This requirement is denoted as *minimal commitment* [EPR08] or *ease of development and configuration* [HIMB05] where a context management framework should expose minimal technical restrictions or constraints for its adoption and integration into existing system infrastructures so that it can be deployed on and utilized by a multitude of different devices. It should be supported by the use of open standards and standard technologies.

- *Sensor abstraction.* Due to the non-existence of a general model on acquiring and processing context (cf. [DAS01, BC04, Dou04, BDR07, Teo08]) existing approaches suffer from the hard-wired integration of sensor and driver logic into application code where sensor details need to handled within the application logic. Application developers and software engineers have to deal with context acquisition and processing in proprietary manners. Decoupling context acquisition technologies from context processing and the application logic by imposing a layered or middleware-based architecture offers greater flexibility and

portability to a context management framework since sensor logic and API details are encapsulated and wrapped in dedicated components that offer common interfaces for their utilization. This requirement is also denoted as *separation of concerns* [DAS01].

- *Multi-granular interpretations.* Context interpretation denotes the task of deriving valuable information from low-level and raw sensorial data and serves as a prerequisite for transforming these data into high-level context information (cf. Section 2.4.4), i.e., creating qualitative assertions about context-relevant aspects using elements of a context ontology. A context management framework therefore must support and allow for different interpretations of contextual information in a transparent manner. Due to the fact that identical information contained in context descriptions is interpreted and handled differently by consuming applications, context descriptions must be reusable by those applications. The integration of additional layers for context aggregation and refinement in the interpretation process allows for deriving qualitative and high-level assertions from numerical observables (cf. [CCDG05]).

- *Flexible communication infrastructure.* A context management framework must employ a communication infrastructure that abstracts over the concrete location and protocol details a context source exposes. One possible solution is to use dedicated components called *context providers* ([BKL+08a, CRL+09, ZS10]) or *sentient objects* ([BC04]) to wrap context sources that utilize a communication infrastructure to exchange contextual information with a context framework; for instance, they map requests issued by the context framework or context consumers to sensor APIs and vice versa and use different technologies such as Remote Method Invocation (RMI), HTTP, SOAP, or other communication protocols to communicate with a framework. For distributed context sources such as Web services or remote repositories, this task is more fragile and requires the reliance on common and standardized communication and transport protocols.

- *Persistence and constant availability of contextual information.* Acquired context data must be constantly available to context consumers. Due to the fact that most of the recently proposed context acquisition architectures are built on middleware technology and are designed for distributed and ubiquitous environments, contextual data acquired in a distributed manner need to be persisted until they can be processed and aggregated with data acquired from other context sources. Therefore, efficient mechanisms for context storage and context query are necessary since the time when a context consumer issues a request for contextual data is in general unknown.

- *Ad-hoc integration of context sources.* This requirement is not discussed in detail in this work as the ad-hoc integration of context sources during run-time requires the support of application-specific context models to allow for *dynamic context service discovery* [WX06] regardless of technical constraints and concrete applications operating on top of a context framework. However, we assume that the context sources used for data replication tasks are unlikely to change very frequently wherefore we will not treat this requirement in detail in the course of this thesis; instead, it is subject to future research.

A context-sensitive RDF data replication framework targeted towards mobile platforms must incorporate most if not all of these requirements to deal with unpredictably changing user contexts and tasks, and the ad-hoc character of mobile environments as well as the increasing error-proneness of information systems operating in such environments. In particular, special emphasis has to be given to the requirements of *minimal platform dependency* and *sensor abstraction* due to the diversity of mobile platforms and mobile operating systems as well as

the heterogeneity of locally deployed sensors and their APIs. The conceptual system architecture and underlying abstract processing models presented in the following sections have therefore been designed on a generic basis regardless of concrete technological components, although the beneficial effects of deploying semantic technologies for the management and processing of contextual information are acknowledged, primarily in more recent approaches (e.g., [WZGP04, HMD05, EPR08, BKL+08a, ZS10]) and also emphasized in Chapter 2 in this work. In particular in relation to the requirements of employing *open and flexible context representation models* and providing support for the *multi-granular interpretation* of contextual information, Semantic Web technologies facilitate the interoperability between autonomously and independently operating context processing systems[5], where created context models are mutually exchanged in an ad-hoc manner; comparable approaches are presented and discussed in, e.g., [RSP07, HDW09]. As mobile devices are operated in different contexts and acquire a multitude of different context-relevant data, the previously mentioned requirements are particularly relevant for the context-sensitive RDF data replication framework proposed in this work. The framework has been specifically designed for direct deployment on mobile devices and to allow for an independent and autonomous acquisition, management, storage, and dissemination of contextual information regardless of application-specific or client-server-based architectures (cf. [BB08, BKL+08a, WRS+05]); its main goals can be summarized as follows:

- *To provide a storage repository for semantic data on a mobile device.* With the increasing proliferation of services based on Semantic Web technologies, the need for mechanisms to store, manipulate, and retrieve RDF data on mobile devices becomes apparent. The local storage of RDF data on a mobile device not only reduces the dependency on a permanent network connection, but also enables the implementation of more efficient search and reasoning algorithms, and extends the user's local information space.
- *To make efficient use of available context information.* Modern mobile devices provide a magnitude of options to capture the user's context, which can be used to infer future information needs and adapt application and device behavior. A semantically appropriate interpretation of these context data helps to build more user-oriented applications and services and enhance the overall mobile user experience.
- *To proactively provide context-relevant data on the device.* As stated before, we cannot rely on a permanent network connection in mobile scenarios. On the other hand, we can infer future information needs from the user's current context information and thus proactively retrieve data from remote data sources to the mobile device that might become relevant in the future, and buffer it using the local storage repository.
- *To provide the technical infrastructure for high-level context processing.* The dynamic and flexible characteristic of our context framework enables the deployment of additional high-level context recognition and utilization services on mobile devices to enable situation awareness (cf. [ATH07, Geh08, LFWK08, SWB+08, THS09]). The framework facilitates almost all aspects of a mobile context processing and management architecture and serves as a foundation for the systematic management and exchange of context descriptions using open semantic standards.

For the realization of these goals, we synthesized concepts from graph theory, distributed transaction management, and the Semantic Web to build an architectural infrastructure for mobile

[5] The term *system* here is used instead of framework to indicate and denote a broader set of context producing and context consuming components.

systems that combines context acquisition, context management, and data replication tasks to replicate data related to the user's current and future information needs in a transparent and proactive manner. Our approach exhibits two significant advantages compared to existing server-based approaches:

(i) Contextual information is acquired, processed, and disseminated directly on a mobile device and does not depend on the availability of external systems. This reduces security and privacy issues since highly private data such as contact information or appointments do not need to be transferred outside a mobile system. It also serves as a technical infrastructure for the deployment of additional high-level context processing and recognition services.

(ii) Furthermore, the generic structure and system design allows for applying the framework to a wide variety of application scenarios and adjusting it to specific replication or data provisioning tasks. For instance, the framework is used in a subsequent project for the proactive provision of patient-related data in the health-care domain.

In the next sections, we give an overview of the conceptual model underlying our framework and we describe the conceptual workflow from acquiring context-relevant information towards the local RDF data replication based on an analysis of the user's current context, storing that data, and making it available to external applications and services.

4.2 Conceptual Context Acquisition and Data Replication Workflow

After having introduced design considerations and requirements of the proposed context-sensitive RDF data replication framework, we now give an overview of the subjacent abstract context acquisition and data replication workflow together with its main conceptual components. The '$\oplus$'-symbol is used to indicate both cooperations between components as well as amalgamations between elements created by specific components; in particular, they exist between context providers as well as between context and orchestration models (cf. Figure 4.1). For the description of the conceptual replication workflow, we do not differentiate between these types of cooperations or amalgamations respectively in more detail, i.e., we use the same symbol to indicate relationships between context providers as well as to indicate an aggregation of context and orchestration models based on a particular logic or scheme; the aggregation of these elements is formally defined in the next section.

We have decided to completely decouple the tasks of context acquisition and data replication (cf. Figure 4.1) to enable a maximum of flexibility w.r.t. the processing of relevant data and adapting the proposed framework to specific application needs. Context-relevant data are retrieved by dedicated components, which we designate *context providers (cp)* in the course of this work and formally define in Section 4.3.3. Context providers act as wrappers around a multitude of different context sources, which might be locally or remotely operating ubiquitous sensors, local applications and data sources, or Web applications respectively Web services[6]. Context providers acquire and collect context-relevant data from external[7] sources, process them, and convert them

[6]More details about the different context sources are provided in Section 4.6.

[7]External in this context is used to denote sources that lie outside the replication framework's computational space, i.e., they operate or are operated independent from the framework in a self-contained manner.

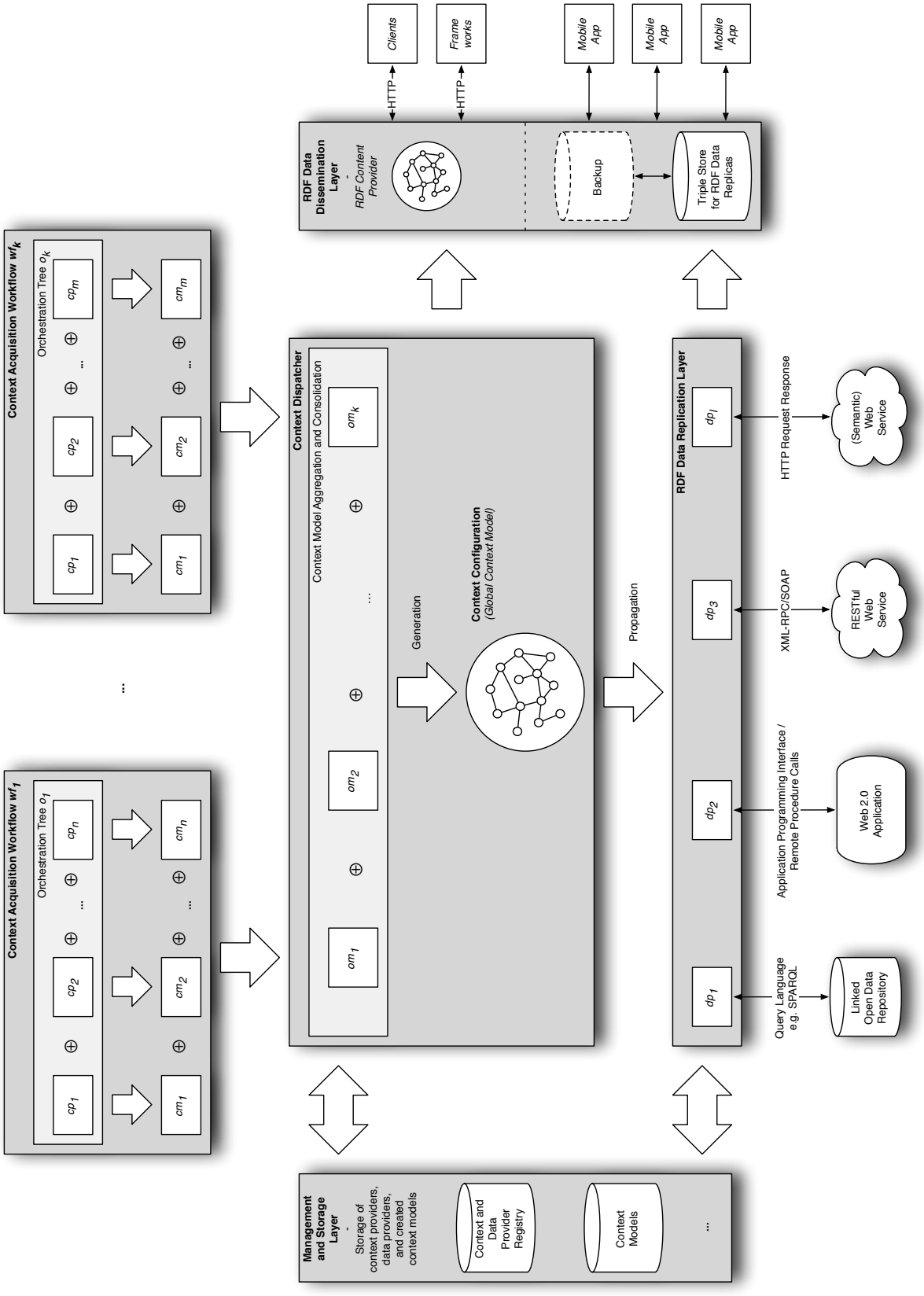


FIGURE 4.1: Conceptual architecture of the components involved in a data replication process

into so-called *context models*, denoted as cm[8] in Figure 4.1, on the basis of well-defined vocabularies and schemas. A context model is a structured and well-defined representation of, in most cases, low-level raw sensor data being described using elements from controlled vocabularies[9]. A formal specification of the acquisition process together with detailed information regarding the involved elements is given in the next section.

In order to augment independently acquired contextual information and to built more sophisticated elaborations of the user's current context, context providers are orchestrated in so-called *orchestration trees*, denoted as o in Figure 4.1, that were derived from a dynamically calculated *orchestration matrix* being created based on a pre-calculated *compatibility matrix* and a particular orchestration logic (cf. Section 4.3.5); in-depth details of the orchestration algorithms, the calculation of both compatibility matrix and orchestration matrix, and the derivation of orchestration trees from the orchestration matrix is given in the Section 4.4 and 4.4.3. Consequently, a prerequisite of the orchestration process is that context providers, per definition, must be able to describe the data they emit and provide information about the contextual information they acquire. To facilitate that process, we have developed a minimal and lightweight data description ontology context providers have to comply with in order to exhibit a structured description of the elements their context models consist of (cf. Section 4.4.1). This information allows for computing several compatibility scores that serve as a basis for computing the compatibility matrix, which by itself serves as the basis for calculating the orchestration matrix and deducing the orchestration trees.

Orchestration trees are static representations of a sequence of context acquisition activities, i.e., they specify how the context models emitted by the containing context providers can be combined in a useful way to facilitate context augmentation and complementation. Those workflow specifications need to be executed in controlled environments to ensure that the context providers being orchestrated in an orchestration tree are executed independently and autonomous from one another and that the replication framework remains in a consistent and deterministic state. This decentralized organization and distributed execution of context acquisition workflows requires a sophisticated execution and control infrastructure and thus has several implications on the conceptual system design, which are discussed in Section 4.6. Therefore, orchestration trees are embedded in so-called *context acquisition workflows*, denoted as wf in Figure 4.1, that are responsible for the controlled execution of technically separated and autonomous but conceptually, i.e., in terms of their context models, related acquisition processes of the context providers embedded within an orchestration tree. In the context of this work, context acquisition workflows are regarded as discrete atomic units that consist of a distinct number of nested transactions and initiate and control the acquisition processes of the context providers orchestrated within the corresponding orchestration trees. They provide the necessary basis for combining and aggregating independently acquired context models, symbolized by the '$\oplus$'-symbol in Figure 4.1. These independently acquired context models are aggregated into a compound context acquisition model called *orchestration model*, being denoted as om in Figure 4.1, which represents an aggregated compound of the context models emitted by the context providers orchestrated in the corresponding orchestration tree.

[8] Please note, that the number used in the subscript of the symbols for both context providers (cp) and context models (cm) indicates a local membership, i.e., two elements sharing the same symbol and subscript are not considered identical as they were integrated in different orchestration trees or context acquisition workflows. The subscripts used in Figure 4.1 hence serve as means to distinguish between elements of the same type that were deployed in the same upper-level element.

[9] By the term 'context model', we refer to a concrete instance of a context model rather than to the underlying scheme that is to be defined by each context provider individually. We provide more details about context models as well as the distinction between context model instances and their underlying prescribing scheme in Section 4.3.2.

The orchestration models by themselves are collected by the *context dispatcher* (cf. Section 4.3.8) and aggregated to a global context model named *context configuration* (cf. Section 4.3.8) that represents a complete description of the user's current context. This global context model is created on demand, i.e., it is created dynamically whenever a new orchestration model has been aggregated from the context models emitted by the context providers orchestrated in an orchestration tree. During the aggregation process, which is performed completely automatically and decoupled from other framework-specific processes, several reasoning and consolidation heuristics are applied to built a more sophisticated and elaborated description of the user's current context. Detailed information about constituting algorithms and processes is given in Section 4.5.

The context configuration is then forwarded to the *data providers*, denoted as *dp* in Figure 4.1, operating in the *RDF data replication layer* where the context configuration is analyzed and used as a basis for replicating RDF data to the device. The data replication layer controls all data providers deployed in an instance of the context framework, which, based on the results of their analysis, initiate and adjust their data replication tasks accordingly. Just as context providers, data providers wrap a specific data source, where no restrictions are impsoed on what a data source might be: a data provider can replicate data from file objects located on the local file system, from databases, from Web services and Web applications, as well as from Linked Data sources exposed on the so-called *Web of Data*[10]. More information about data providers together with a formal model of them are given in Section 4.3.10. The replicated data are stored in a local *triple store* and made available through an *RDF content provider* operating in the *RDF data dissemination layer*. This layer including the RDF content provider and the triple store are described in Section 4.6.

The dissemination layer offers external clients controlled access to both replicated data stored in the local triple store as well as to the context configuration so that context consumers are able to continuously obtain a structured and consolidated representation of the user's current context. This allows other context management frameworks or context-sensitive services to either process the information contained in the context configuration and complement their own context models, as well as to adjust their services based on specific aspects of the user's context thus delivering more user-focused and user-friendly services. The RDF content provider also allows locally operating mobile applications to access and utilize data replicas so that more sophisticated applications and services can be built on top of the user's current context. Detailed information about the locally operating triple store and the dissemination of both replicated data as well as the context configuration will be provided in Section 4.6.2 whereas [ZS12b] exhibits detailed implementation-specific information of the RDF content provider.

A loose, data-based coupling between context providers and data providers is realized through the *context dispatcher*, which is notified every time a context provider detects a change in a context source it observes. The context dispatcher aggregates, consolidates, and reasons on context information, and forwards them to the appropriate data provider components as described before. An algorithmic description of the main tasks and the process model of the context dispatcher is given in Section 4.6.

The *management and storage layer* hosts repositories for both context and data providers as well as for emitted context models and created context configuration instances. Other framework components can access the elements stored in these repositories in a controlled and uniform way where the storage and management layer keeps track of changes applied to the repositories' elements.

[10] According to recent studies related to the information needs of mobile users, an estimated amount of 58% of such information needs can be satisfied through the provision of data replicated from publicly available data sources (cf. [SLGH08]).

4.3 Formal Model

In the previous section, we have informally described the conceptual workflow of acquiring and aggregating contextual information and replicating RDF data being relevant to the user's information needs to the mobile device based on an analysis of a global representation of the user's current context – the context configuration[11]; we also gave some details regarding the characteristics of the involved elements respectively components. In the present section, we now define the main constituting elements of the proposed context-sensitive replication framework on a formal basis using set theory and algebraic specifications and provide profound details regarding their role and characteristics in data replication processes.

Before we start with the definition of the formal symbols, we introduce the concepts of *sensor* and *context source*, which we consider fundamental to this work. We refer to a sensor as being a physical or logical entity that captures raw sensorial data and produces a computational representation together with software events in reaction to a real-world stimulus [BC04]. Sensors are realized as hardware devices or software components, or a combination of both. Putting this definition in a broader, more general context, we can regard sensors as context sources that, from a technical perspective, can be anything ranging from physical sensors deployed in ubiquitous environments, over software or logical sensors (in some works such sensors are called *virtual sensors*), towards Web applications, services, and repositories offering context-relevant or context-complementing information. As the work continues, the term context source is therefore used to denote hardware or software entities that allow for acquiring information that have a specific relationship to the user's current real-world contexts. In consequence, we will refer to the more general concept of context source rather than context sensor in the course of this work.

A fundamental design consideration of the proposed framework is the ability to acquire context-relevant information from a wide variety of different context sources where no restrictions have been made according to what a potential context source might be. As first outlined in the previous section regarding the *operationalization*, that is, the integration of context sources in the framework's computational and processing space and using it for the exploitation of contextual information, a context source needs to be wrapped by a context provider, which offers unified and well-defined interfaces to the APIs exhibit by a context source as well as a structured and well-defined description of the contextual information being emitted by a context source. Following the conceptual description of context providers given in Section 4.2, we employ *two* types of context providers: *primary* (i.e., active) and *complementary* (i.e., passive) context providers. Primary context providers encapsulate a hardware or software sensor and become active whenever a change in a context source is detected, i.e., they actively monitor the status of a context source and autonomously capture sensorial stimuli. Complementary context providers employ a different, reactive operation model; they react according to responses of primary context providers and become active whenever a corresponding compatible primary context provider emits a context model that can be refined or complemented in the course of their acquisition workflows. The main purpose of complementary context providers therefore is to take the context models emitted by primary context providers as input data for initiating their acquisition tasks and thus performing a form of *context augmentation* or *context complementation* (cf. Section 3.4).

In the following, we give a formal definition of the concept of context models and context providers as being defined and understood in the scope of the proposed framework together with other relevant components involved in data replication processes. For the formal definition

[11] As outlined in Section 4.6, the context configuration resembles a consolidated and global representation of the aggregated single context models.

of the involved components, we follow a natural path that leads from the acquisition of contextual information, towards their transformation and aggregation into the context configuration.

4.3.1 Symbols and Relations

We start this section with an introduction of the symbols that are considered relevant for and constitute the formal model underlying the proposed framework. The symbols are introduced in Definition 4.1 and will be thoroughly defined in their respective sub sections. For reasons of clarity and comprehensibility, we summarized all the symbols used to describe the formal model of the proposed context-dependent replication framework in Table 4.1. However, Table 4.1 lists only those symbols that are considered as constituting elements with respect to the formal model; symbols that are defined in the course of a definition of a main constituting element such as the set of raw data tokens Σ, the set of symbolic data tokens Γ, or the set of states Q a context provider can transition into are not included as such symbols serve as auxiliary elements for defining their superordinate symbols. Furthermore, we list the set of relations R existing between selected elements of the formal model in a separate table (Table 4.2) together with a more comprehensive and detailed description. Please note that the formal model contains a number of additional elements not depicted in the conceptual workflow as expounded in the previous section. The purpose of those elements is discussed in detail in the respective sub sections.

Definition 4.1 (Symbols)**.** Let CP denote the set of all context providers $cp \in CP$ currently deployed in a running instance of the proposed framework. Further, let C denote the set of all complementary context providers $c \in C$ and let P denote the set of all primary context providers $p \in P$. Let S denote the set of all context sources $s \in S$ that are wrapped by a context provider $cp \in CP$ irrespectively of their concrete type, and let M denote the set of all context models $m \in M$ emitted by the context providers $cp \in CP$. D denotes the set of context descriptions $d \in D$ that encapsulate a context model $m \in M$ and serve as means for the communication of context models $m \in M$ and for the inclusion of additional information regarding the trustworthiness and quality of the encoded contextual information. Furthermore, let O denote the set of all orchestration trees $o_p \in O$ and let WF denote the set of context acquisition workflows $wf_{o_p} \in WF$ that are responsible for the controlled execution of the context providers $cp \in CP$ orchestrated in the orchestration trees $o_p \in O$. In this respect, let M_{o_p} denote the compound and aggregated set of all context models $m \in M$ of the context providers $cp \in CP$ orchestrated in an orchestration tree $o_p \in O$ and OM the entire set of orchestration models $M_{o_p} \in OM$. For convenience reasons, let C_p denote the set of complementary context providers $c \in C$ that are orchestrated in an orchestration tree $o_p \in O$ and R_p the set of relations that exists between them. In addition, let $\mathbf{M}^O$ denote the orchestration matrix that serves as a basis for building the orchestration trees $o_p \in O$ and $\mathbf{M}^C$ the compatibility matrix that records the compatibility scores computed between all $cp \in CP$ on the basis of their data descriptions *ddesc*. Finally, let CC denote the set of all created context configurations $cc \in CC$ and DP the set of all data providers $dp \in DP$.

Based on these conventions, we can define the overall set of context providers CP as being the union of the set of primary context providers P and the set of complementary context providers C with $CP := P \cup C$ where both sets P and C are disjunct. Based on the framework's configuration, the sets CP and P can be identical with $CP = P$ in case only primary context

Table 4.1: Symbols used in the formal model

Symbol	Description
cp	General symbol for a context provider irrespective of its type
p	Primary context provider
c	Complementary context provider
s	Context source being wrapped by a context provider $cp \in CP$
m	Context model emitted by a context provider cp in form of a context description
d	Context description that wraps a context model m
$ddesc$	Data description that specifies the vocabulary elements a context provider cp uses for describing the context model m it emits and requires
dp	Data provider
CP	Set of all context providers cp deployed in a running instance of the framework
P	Set of all primary context providers p
C	Set of all complementary context providers c
S	Set of all context sources s wrapped by a context provider $cp \in CP$
D	Set of all context descriptions d
DP	Set of all data providers dp
M	Set of all context models m emitted by context providers $cp \in CP$
$\mathbb{M}$	Set of all possible combinations of asserted RDF statements that can be created from the set of RDF resources $\mathbb{U}$, blank nodes $\mathbb{B}$, and literals $\mathbb{L}$
M_p	Set of context models m emitted by the primary context provider $p \in P$
M_c	Set of context models m emitted by the complementary context provider $c \in C$
o, o_p	General symbol for orchestration trees; the subscript o_p indicates the correspondence to the primary context provider $p \in P$ an $o \in O$ refers to
O	Set of all orchestration trees o_p corresponding to the primary context providers $p \in P$
V_p	Set of context providers $cp \in CP$ that constitute an orchestration tree $o_p \in O$ that corresponds to a primary context provider $p \in P$
C_p	Set of complementary context providers $c \in V_p$ that are orchestrated in an orchestration tree $o_p \in O$ that corresponds to a primary context provider $p \in P$
R_p	Set of relations r that exist between the context providers $cp \in V_p$ orchestrated in an orchestration tree $o_p \in O$
$\delta^+_{o_p}(cp)$	Set of relations $r \in R_p$ that exist between cp and its adjacent context providers cp_{succ} such that cp_{succ} is positive incident to cp over r with $cp \neq cp_{succ}$
$\delta^-_{o_p}(cp)$	Set of relations $r \in R_p$ that exist between cp and the preceding context provider cp_{prec} such that cp is positive incident to cp_{prec} over r with $cp_{prec} \neq cp$
$N^+_{o_p}(cp)$	Set of complementary context providers $c \in C_p$ that are adjacent to a context provider $cp \in V_p$ for $cp \neq c$ and positive incident to cp over a set or relations $r \in R_p$
$N^-_{o_p}(cp)$	Set of context providers $cp_j \in Vp$ to which a context provider $cp \in V_p$ is adjacent to for $cp \neq cp_j$
M_{o_p}	Compound context acquisition model of all context providers orchestrated in an orchestration tree $o_p \in O$
OM	Set of all compound context acquisition models M_{o_p}
wf, wf_{o_p}	General symbol for a context acquisition workflow, wf_{o_p} indicates the orchestration tree a context acquisition workflow pertains to
WF	Set of all context acquisition workflows wf
cc	Context configuration
CC	Set of all context configurations cc
Π	Set of reasoning rules for creating a context configuration $cc \in CC$
$\mathbf{M}^C$	Compatibility matrix containing numerical values that indicate the compatibilities between all context providers $cp \in CP$
$\mathbf{M}^O$	$\lvert CP\rvert \times \lvert CP\rvert$ orchestration matrix that records adjacency relationships between all $cp \in CP$ and serves as a basis for deducing the orchestration trees $o_p \in O$
$\mathbf{M}^A_{o_p}$	$\lvert V_p\rvert \times \lvert V_p\rvert$ adjacency matrix of an orchestration tree $o_p \in O$

TABLE 4.2: Relations existing between selected elements of the formal model

Relation	**Description**
$R_{emitted} : CP \times M$	is defined as the injective relation between the elements $cp \in CP$ and the context model instances $m \in M$ they emit. A context provider $cp \in CP$ is able to emit several instances of context models $m \in M$ whereas a context model $m \in M$ per definition corresponds to only one distinct context provider $cp \in CP$.
$R_{corresponds} : P \times O$	defines the bijective relationship between the primary context providers $p \in P$ and $o_p \in O$ as for every primary context provider $p \in P$ there exists exactly one $o_p \in O$. To indicate this relationship, we attach the name of a primary context provider $p \in P$ to its corresponding orchestration tree such that it is denoted as o_p.
$R_{executedBy} : O \times WF$	is defined as the bijective relation existing between an orchestration tree $o_p \in O$ and the context acquisition workflow $wf_{o_p} \in WF$ assigned to an orchestration tree $o_p \in O$ that controls and monitors the acquisition workflows of the context providers $c \in C$ being orchestrated in it.
$R_{compose} : O \times OM$	is defined as the bijective relation existing between an orchestration tree $o_p \in O$ and the compounded context acquisition models $M_{o_p} \in OM$ of its containing context providers $p \in P$ and $c \in C$.
$R_{convey} : D \times M$	is defined as an bijective relation that exists between a context description $d \in D$ and the context model $m \in M$ emitted by a context provider $cp \in CP$.

providers $p \in P$ are deployed in a framework's running instance whereas C is always a real subset of CP since complementary context providers $c \in C$ per definition require at least one primary context provider $p \in P$ to be operational. This aspect, among others, is formally specified in Section 4.3.5 expounding the orchestration logic underlying an orchestration tree. The dependencies existing between the sets CP, C, and P are expressed in Equation (4.1):

$$CP := P \cup C \quad \text{with} \quad P \subseteq CP \wedge C \subset CP \quad \text{and} \quad P \cap C = \emptyset \tag{4.1}$$

Furthermore, we define the relations listed in Table 4.2 as being existent between the different elements constituting the formal model. Those relations represent dependencies and correspondences between selected elements of the formal model where the specific semantic of these relations is discussed in detail in the sections of the respective elements they refer to. Moreover, since there exist distinct 1:1 relations between the sets P, O, OM, and WF, the corresponding relations $R_{corresponds}$, $R_{executedBy}$, and $R_{compose}$ are *transitive* such that for the relation $(p_i, o_{p_i}) \in R_{corresponds}$ and $(o_{p_i}, wf_{o_{p_i}}) \in R_{executedBy}$, we can deduce due to the *transitive closure* of these relations that there exists a relation $(p_i, wf_{o_{p_i}})$ between a primary context provider $p_i \in P$ and a context acquisition workflow $wf_{o_{p_i}} \in WF$. In the same way, there exists an implicit bijective relation $(wf_{o_{p_i}}, M_{o_{p_i}})$ deduced from the transitive closure of the relations $(o_{p_i}, wf_{o_{p_i}}) \in R_{executedBy}$ and $(o_{p_i}, M_{o_{p_i}}) \in R_{compose}$. Since those relations can be deduced due to the transitivity of existing relations defined in the course of the formal model, the two aforementioned relations have not been explicitly defined in Table 4.2.

Prior to defining a context provider $cp \in CP$, we give a formal definition of a context model m as this definition will be used in nearly all subsequently following definitions.

4.3.2 Context Model

As we emphasize the use of Semantic Web languages and technologies for the representation of contextual information, we define a context model $m \in M$ in technology-specific terms as being an RDF graph consisting of a finite set of resources represented by their URIs and denoted as $\mathbb{U}$, a finite set of blank nodes $\mathbb{B}$, and a finite set of literals $\mathbb{L}$[12]. Those elements are the building blocks of RDF statements, which we have formally defined in Definition 2.2. As a consequence, a context model $m \in M$ itself is an element from the power set $\mathbb{M}$, which is defined as the cartesian product of the union of $\mathbb{U}$ and $\mathbb{B}$ referring to the *subject* of an RDF statement, $\mathbb{U}$ referring to a statement's *predicate*, and the union of the sets $\mathbb{U}$, $\mathbb{B}$, and $\mathbb{L}$ that refer to the *object* of an RDF statement. We therefore define a context model $m \in \mathbb{M}$ as follows:

Definition 4.2 (Context Model)**.** Let m denote a context model created by a context provider $cp \in CP$ and let M denote the set of all context models $m \in M$ emitted by the context providers $cp \in CP$ in the course of their acquisition workflows irrespectively of their concrete type. A context model m thus is an element from the powerset $\mathbb{M}$ that is defined as the union of all possible context models that can be created from combining the set of URIs $\mathbb{U}$, blank nodes $\mathbb{B}$, and literals $\mathbb{L}$ according to the RDF data model structure (cf. [Bec04]) and semantics (cf. [HM04]) as expressed in Equation (4.2):

$$m \in \mathbb{M} \quad \text{where} \quad \mathbb{M} := \mathcal{P}(\mathbb{U} \cup \mathbb{B} \times \mathbb{U} \times \mathbb{U} \cup \mathbb{L} \cup \mathbb{B}) \tag{4.2}$$

Please note that the term '*context model*' refers to a concrete instance of a context model rather than to the underlying scheme employed by a context provider $cp \in CP$ to create a context model instance it emits. We use and understand the term '*scheme*' in this thesis as it was defined in the Linked Data context (cf. [HB11]) denoting the act of combining distinct elements (terms) defined externally to the data source making use of such elements in order to describe the data to be published by a data provider[13]. This definition perfectly applies to our concept of context providers as they exactly operate according to the Linked Data principles (cf. [BL06a, BHBL08]) by providing a structured representation of excerpts of the universe of discourse that are described with elements from well-defined, well-understood, and well-established semantic vocabularies whose elements are specified in an axiomatic way using the Semantic Web languages, RDF, RDFS, OWL, and OWL2. In consequence, we do not specify a general context model scheme in the course of this work since this falls into the responsibility of each context provider $cp \in CP$ separately and will offer a maximum of flexibility in representing, exchanging, and integrating contextual information.

In addition to the definition of the set M containing all emitted context models $m \in M$, we define partitions of M denoted as M_p or M_c for every primary context provider $p \in P$ and complementary context provider $c \in C$ depending on the type of context provider a partition refers to. Hence, a partition M_{cp} contains only those context models $m \in M$ that have been emitted by the corresponding context provider $cp \in CP$. In consequence, the set M_{p_i} for instance corresponds to a specific primary context provider $p_i \in P$ and contains only those context models $m \in M$ specifically emitted by the context provider $p_i \in P$. All sets M_p and M_c of all context

[12] For the purpose of this model, no distinction is made between plain and typed literals (cf. [KC04]) as these distinction is irrelevant for the formal definition of the context models $m \in \mathbb{M}$.

[13] The phrase 'data provider' is not to be confounded with the concept of a data provider as introduced and defined in this work.

providers $cp \in CP$ are thus partitions of the set of all emitted context model $m \in M$ such that

$$\left(\bigcup_{p \in P} M_p \right) \cup \left(\bigcup_{c \in C} M_c \right) = M. \tag{4.3}$$

In order to relate a context model $m \in M$ to the context provider $cp \in CP$ it has been created and emitted by, we explicitly express this relationship through the binary and injective relation $R_{emitted}$ (cf. Table 4.2) existing between the set of context providers CP and the set of emitted context models M such that $R_{emitted} \subseteq CP \times M$. A context provider $cp \in CP$ can emit multiple context models $m \in \mathbb{M}$ whereas a context model $m \in M$ per definition only corresponds to one specific context provider $cp \in CP$ that was responsible for its creation. This fact allows us to define the sets M_p and M_c more precisely:

$$M_p := \left\{ m \in M \mid \exists! \, p \in P : (p, m) \in R_{emitted} \right\} \tag{4.4}$$

$$M_c := \left\{ m \in M \mid \exists! \, c \in C : (c, m) \in R_{emitted} \right\} \tag{4.5}$$

These definitions allow us to state that for any element $m \in M_p \vee m \in M_c$ there exists a relation $(p, m) \in R_{emitted}$ or $(c, m) \in R_{emitted}$ between the corresponding primary respectively complementary context providers $p \in P$, $c \in C$ and a context model $m \in M$ such that

$$\forall p \in P \, \exists! \, M_p \subseteq M : m \in M_p \iff (p, m) \in R_{emitted}. \tag{4.6}$$

In the same way, we can adapt Equation (4.6) for the sets M_c for all $c \in C$:

$$\forall c \in C \, \exists! \, M_c \subseteq M : m \in M_c \iff (c, m) \in R_{emitted} \tag{4.7}$$

Let further define two functions $\phi_{current} : M \longrightarrow M$ and $\phi_{previous} : M \longrightarrow M$ which return the most recently emitted context model $m \in M$ respectively the penultimately emitted one. This distinction is important as it is the basis for establishing compensation strategies to circumvent undefined states and indeterministic behavior caused by malfunctioning or temporarily unavailable context sources $s \in S$ respectively context providers $cp \in CP$ that wrap a context source $s \in S$. Both functions are defined for any set of context models and hence also apply to all partitions of M.

After having defined a context model $m \in M$ emitted by a context provider $cp \in CP$ on a formal basis together with the set of relations existing between the constituting elements respectively sets, we now define a context provider $cp \in CP$ in more detail. Before we give a formal definition of the concept of context provides as they are used and understood in the context of this work, we provide some general information about context sources $s \in S$ as well as the methods related to how context-relevant data can be acquired from them.

4.3.3 Context Provider

Context sources in general employ different operation modes that influence the way how context-relevant data can be retrieved from them. Most common approaches exhibit a push- or pull-based mechanism, where a push-based mechanism informs its clients either on the basis of regular time intervals or when a certain delta in terms of the values it offers is exceeded. For instance, a location-based sensor $s^{location} \in S$ may either provide its position in pre-defined time intervals

or when a specific distance has been covered since its last notification; a temperature sensor $s^{temperature} \in S$ may notify its clients only when the difference between the current temperature $\mathcal{T}(t_n)$ measured at time t_n and the previously communicated temperature $\mathcal{T}(t_{n-1})$ measured at time t_{n-1} is greater than a predefined constant $\Delta\mathcal{T}$ such that $\left|\mathcal{T}(t_n) - \mathcal{T}(t_{n-1})\right| > \Delta\mathcal{T}$. In a pull-based communication, the context consumer initiates the request to a context source $s \in S$, which could be performed on basis of regular time intervals or on demand. However, as such notification mechanisms are implementation-specific and thus being irrelevant for the formal definition of context providers $cp \in CP$, we do not elaborate on them specifically in the course of this chapter. Instead, we base our definitions on the *values* that are acquired from a context source $s \in S$ irrespectively of the concrete technological realization of the communication protocol and mechanism used to gather such data. Due to the diversity of the context data space, we rather suggest to use the notion of data *tokens* to refer to contextual data since a context-relevant aspect does not necessarily have to be represented through distinct numerical values, while the notion of tokens resembles a broader spectrum of potential information entities.

This perception finds expression in the definition of primary and complementary context providers (cf. Definition 4.3 and Definition 4.4), which we define on the basis of automata theory, since we perceive context providers $cp \in CP$ as *transducers* that transform the raw data values emitted by a context source $s \in S$ into a structured and well-defined representation using vocabularies and technologies from the Semantic Web. In contrast to fundamental automata theory (cf. [RS97]) where such raw data values are considered as elements σ from an *input alphabet* Σ and hence are processed as a stream of characters $\sigma_0\sigma_1 \ldots \sigma_n \in \Sigma^*$ constituting an input word $\omega \in \Sigma^*$, a context provider $cp \in CP$ is capable of collecting and processing such raw data values as a whole, and performing additional interpretation, transformation, and aggregation steps. We denote such raw data values acquired from a context source $s \in S$ as *context data tokens*.

As a consequence, we define a context provider $cp \in CP$ on the basis of a *non-deterministic finite state transducer (NFST)* that transforms a defined set of raw data tokens being mostly sensorial values acquired from the context source $s \in S$ it encapsulates into a context model $m \in \mathbb{M}$. Non-deterministic finite state transducers are a special field of automata theory that enable a formal description and a systematic analysis of deterministic and non-deterministic state models. As opposed to deterministic finite state machines, so-called *acceptors*, which generate binary responses (in most cases on the basis of a boolean value space) in reaction to an input word $\omega \in \Sigma^*$ signaling either the acceptance or rejection of the input word ω, transducers transform a defined set of input words using a deterministic or non-deterministic state model and a predefined logic into output words consisting of elements defined in an *output alphabet*. In automata theory, a transducer is defined as an abstract automaton or machine that reads a sequence of characters from an input tape and, based on an analysis of these characters, generates a sequence of output characters to be printed on the output tape. An essential aspect of transducers as defined in automata theory is that the length of the input tape is equal to the length of the output tape, i.e., the amount of characters to be read from the input tape is equal to the amount of characters written to the output tape. However, this declaration does not comply with our definition of context providers as we base our definition on the inclusion of ε-transitions that allow for a more flexible specification of the incorporated processing logic and the transitions between specific states (cf. Table 4.3). We consider that a necessary extension of our definition as the operation environments from which context providers $cp \in CP$ obtain their data are more divergent and heterogeneous compared to closed or controlled environments. The inclusion of ε-transitions allows for defining state transitions not being caused by the reading or by the retrieval of a raw data token $\sigma \in \Sigma$ respectively, but rather by the completion of a specific phase in the acquisition process (e.g. aggregation of symbolic data tokens).

After having expounded the necessary basis on which context providers are perceived in the context of this work, we now provide a formal definition that is based on the formal definition of non-deterministic finite state transducers as defined in automata theory. This definition applies to both types of context providers in the same way even though we extend Definition 4.3 when we provide a formal definition of complementary context providers $c \in C$ as they require an input context model $m \in M$ in addition for performing their acquisition tasks.

Definition 4.3 (Context Provider)**.** A context provider $cp \in CP$ is defined on the basis of a *non-deterministic finite state transducer (NFST)* being the 8-tuple $(Q, \Sigma, \Gamma, \delta, \lambda, \mu, q_0, ddesc)$ that transforms accepted raw data tokens $\sigma \in \Sigma$ acquired from a context source $s \in S$ a context provider wraps into a structured description $m \in \mathbb{M}$ of the context-relevant aspect a context source $s \in S$ represents. A context provider $cp = (Q, \Sigma, \Gamma, \delta, \lambda, \mu, q_0, ddesc)$ consists of the following elements:

- Q is defined as a finite set of states q a context provider $cp \in CP$ transitions into during the process of acquiring contextual information and the transformation of those information into a context model $m \in \mathbb{M}$.
- Σ is the finite set of raw data tokens $\sigma \in \Sigma$ retrieved from a context source $s \in S$ that are being accepted by a context provider $cp \in CP$ with $\varepsilon \notin \Sigma$. This set corresponds to the *input alphabet* of state machines as defined in automata theory.
- Γ is the finite set of symbolic context data tokens $\gamma \in \Gamma$, the so-called *output alphabet* in automata theory, that contains the *transformed* raw data tokens $\sigma \in \Sigma$ and hence serves as a basis for creating a context model $m \in \mathbb{M}$ that a context provider $cp \in CP$ emits.
- $\delta : Q \times \Sigma^* \cup \{\varepsilon\} \longrightarrow 2^Q$ is a surjective state transition function between the finite set of states Q and streams of accepted raw data tokens $\sigma \in \Sigma$ a context provider $cp \in CP$ is capable to process.
- $\lambda : Q \times \Sigma^* \cup \{\varepsilon\} \times Q \longrightarrow \Gamma^*$ is a transformation function that transforms streams of raw data tokens $\sigma \in \Sigma$ into streams of symbolic context data tokens $\gamma \in \Gamma$.
- $\mu : Q \times \Gamma^* \cup \{\varepsilon\} \longrightarrow \mathbb{M}$ defines a transformation and output function that transforms streams of symbolic context data tokens $\gamma \in \Gamma$ being created by a context provider $cp \in CP$ into a context model $m \in \mathbb{M}$.
- $q_0 \in Q$ is the initial state of a context provider $cp \in CP$ after its instantiation and initialization.
- *ddesc* is defined as the description of the data, a context provider $cp \in CP$ needs as input for initiating its acquisition tasks or emits as output data. A data description *ddesc* defines the vocabularies and vocabulary elements (terms) a context provider makes use of for expressing its context model $m \in M$.

Please note that we do not define a separate finite set of final states $F \subseteq Q$ that can be found in some definitions in the related literature since both deterministic and non-deterministic finite state transducer per definition do not require the explicit definition of such a set F as their

primary objective lies in the production of output symbols[14] rather than reaching a final state $q \in F$.

Definition 4.3 defines a context provider on a general basis and formally specifies those aspects that apply to both types of context providers. For complementary context providers $c \in C$, this definition needs to be slightly adapted in order to take into account the aspect that complementary context providers are responsible for refining or complementing contextual information and initiate their acquisition tasks as a result of the existence of a context model $m \in M$ emitted by a context provider $cp \in CP$ to which a complementary context provider $c \in C$ is compatible to. We therefore define a complementary context provider $c \in C$ as follows:

Definition 4.4 (Complementary Context Provider)**.** A complementary context provider $c \in C$ is defined as the 9-tuple $(Q, \Sigma, \Gamma, \delta', \lambda', \mu', q_0, ddesc, m^{input})$ where

- $Q, \Sigma, \Gamma, q_0, ddesc$ refer to the same symbols as defined in Definition 4.3.
- δ', λ', μ' correspond to δ, λ, μ defined in Definition 4.3 but in contrast perform transition, transformation, and output functions relative to the information contained in the input context model m^{input} emitted by a compatible context provider $cp \in V_p$ for $c \neq cp$.
- m^{input} represents the context model m emitted by a compatible context provider $cp \in V_p$ that serves as input data for initiating the context acquisition process of a compatible complementary context provider $c \in V_p$ for $c \neq cp$ in order to refine and complement the information represented in m^{input}.

The initiation of the acquisition processes of complementary context providers $c \in V_p$ is governed by the orchestration logic (cf. Section 4.3.5) underlying the orchestration trees $o_p \in O$ in which context providers $cp \in V_p$ are orchestrated. A complementary context provider $c \in V_p$ becomes active whenever a compatible context provider $cp \in CP$ emits a context model $m \in \mathbb{M}$ to which c is adjacent to for $c \neq cp$.

A context provider $cp \in CP$ usually starts in a state $q_0 \in Q$ before it is able to receive raw data tokens $\sigma \in \Sigma$ from a context source $s \in S$. With every raw data token $\sigma \in \Sigma$ or sequence of raw data tokens $\omega \in \Sigma^*$ received from a context source $s \in S$, a context provider $cp \in CP$ proceeds from its current state $q_i \in Q$ to a subsequently following, i.e., succeeding state $q_{i+1} \in Q$ being an element from the set 2^Q defined by the transition function $\delta(q_i, \omega)$. As a consequence, for every sequence of raw data tokens $\omega \in \left(\Sigma^* \cup \{\varepsilon\}\right)$ and initial state q_i, there exists a number of potentially eligible following states $q_{i+1} \in 2^Q$ such that

$$q_{i+1} \in \delta(q_i, \omega) \quad \text{where} \quad \omega \in \left(\Sigma^* \cup \{\varepsilon\}\right) \tag{4.8}$$

In case all the data tokens $\sigma \in \Sigma$ have been acquired from a context source $s \in S$, the processing logic underlying a context provider $cp \in CP$ can initiate a transition to another state $q_{i+1} \in 2^Q$ by sending the ε element that causes, for instance, the aggregation of the transformed context-relevant data tokens as represented by the state q_4 (cf. Table 4.3) with $\delta(q_1, \varepsilon) \longrightarrow q_4$. As known from automata theory, ε-transitions allow for defining transitions from a state $q_i \in 2^Q$ to a state $q_{i+1} \in 2^Q$ independently of receiving and processing an element $\sigma \in \Sigma$ from a context

[14] According to our definition of context providers, the aggregation and analysis of such output symbols $\gamma \in \Gamma$ form a context model $m \in \mathbb{M}$ as opposed to automata theory where such output symbols written to the output tape form an output word $\omega = \gamma_0\gamma_1 \ldots \gamma_n$ where $\omega \in \Gamma^*$.

TABLE 4.3: List of the generic states $q \in Q$ of a context provider $cp \in CP$

State	Phase	Description
q_0	`Initializing`	In this state, a context provider $cp \in CP$ initiates the necessary steps for establishing a connection to the context source $s \in S$ it wraps and retrieves data from.
q_1	`Receiving`	In q_1, a context provider $cp \in CP$ is ready to receive context data tokens $\sigma \in \Sigma$ emitted by a context source $s \in S$. Depending on the underlying processing logic, a context provider $cp \in CP$ might capture a sequence of context data tokens $\omega = \langle \sigma_0, \sigma_1, \ldots, \sigma_n \rangle \in \Sigma^*$ before it transitions to a state q_2.
q_2	`Transforming`	When a context provider $cp \in CP$ received data from a context source $s \in S$, this data $\sigma \in \Sigma$ is transformed to symbolic values $\gamma \in \Gamma$ that a context provider $cp \in CP$ can interpret and process in subsequently following steps q_{2+i}.
q_3	`Clustering`	Clustering allows for buffering multiple elements $\gamma \in \Gamma$ in case a context provider $cp \in CP$ supports the aggregation of, e.g., multiple elements from a context data stream $\omega = \langle \gamma_0, \gamma_1, \ldots, \gamma_n \rangle \in \Gamma^*$.
q_4	`Aggregating`	During the aggregation state, a context provider $cp \in CP$ analysis the symbolic values acquired from a context source $s \in S$ and transforms them into a more consolidated representation while conducting data cleansing steps.
q_5	`Reasoning`	When the symbolic data have been cleansed, a context provider $cp \in CP$ can apply an individual set of reasoning heuristics in order to augment the acquired data and to deduce relationships between data elements not explicitly represented.
q_6	`Creating`	In the creating state, the internal representation of the acquired symbolic data tokens is transformed into a context model $m \in \mathbb{M}$ based on explicitly defined semantic vocabularies and a well-defined schema employed by a context provider $cp \in CP$.

source $s \in S$. Through the inclusion of ε-transitions, the context acquisition and processing logic underlying context providers can be realized in more flexible ways as their context acquisition and processing logic is defined on a less rigid basis without loosing expressiveness or processing capabilities.

However, as a supplement to the definitions of context providers, we define a set of general states $q \in Q$ defined in Table 4.3 that universally apply to context providers $cp \in CP$ irrespectively of their concrete type as well as of the type of context source $s \in S$ being wrapped. Those states $q \in Q$ represent the different phases a context provider $cp \in CP$ can transition into in the course of acquiring contextual information and transforming those information into a context model $m \in \mathbb{M}$. They were derived from an analysis of the relevant literature regarding context acquisition and context management reference architectures as discussed in Section 2.4.3. The set of general states $q \in Q$ deduced from related reference models can be extended or adapted as needed depending on whether or not a concrete context provider $cp \in CP$ provides support for a specific aspect or functionality such as context data clustering. We have omitted states related to failures or malfunctions of context providers $cp \in CP$ respectively the context sources $s \in S$ they wrap for reasons of clarity and comprehensibility.

Table 4.3 is to be read as follows: when a context provider $cp \in CP$ is in a state q_i, for instance in q_0, it currently performs all operations necessary for its initialization; if these operations are completed successfully, it moves on to a state q_{i+1}. According to the initialization example, the sequentially following state is q_1 where the context provider is now ready to receive data from the

context source $s \in S$ it encapsulates. Please note that we define the formal model independently from a concrete communication mechanism and protocol as these are implementation-specific and depend on the concrete realization of a context source. In case a context provider $cp \in CP$ receives a specific sequence of raw data tokens $\omega \in \Sigma^*$ or its internal logic[15] initiates a transition to another state $q_{i+1} \in Q$ based on the positive evaluation of a specific transition condition (which might also be context provider-specific), the context provider $cp \in CP$ shifts to the state q_2 in which the recently received raw data tokens $\sigma \in \Sigma$ are transformed into symbolic context data tokens $\gamma \in \Gamma$ so that they can be interpret by a context provider and further processed in subsequent steps. When a raw data token $\sigma \in \Sigma$ is to be transformed into a symbolic data token $\gamma \in \Gamma$, this token might then be buffered for context clustering (cf. the TEA architecture as introduced by [GSB02]), wherefore a context provider $cp \in CP$ transitions into the state $q_3 \in Q$ to better analyze acquired context-relevant data according to specific patterns, e.g., time-dependent transitions in context data. A context data cluster can then be aggregated into a more consolidated representation while conducting data cleansing steps to better infer on present context data patterns, which is performed in state $q_4 \in Q$. Optionally, a context provider $cp \in CP$ might apply a set of individual reasoning heuristics or reasoning rules to infer additional information and relationships not explicitly asserted in the contents of the consolidated representation of context data. These steps are performed when a context provider $cp \in CP$ is in a state $q_5 \in Q$. In the creating state $q_6 \in Q$, the consolidated (internal) representation is transformed into an RDF-based representation, the context model $m \in \mathbb{M}$, that is an element from the power set $\mathcal{P}$ of the vocabulary elements defined in the data description *ddesc* of the context provider $cp \in CP$ with $\mathcal{P}\big(\mathbb{U}_{cp} \cup \mathbb{B}_{cp} \times \mathbb{U}_{cp} \times \mathbb{U}_{cp} \cup \mathbb{L}_{cp} \cup \mathbb{B}_{cp}\big) \subseteq \mathbb{M}$[16]. The transformation of the internal symbolic context data tokens into a context model $m \in \mathbb{M}$ is conducted on the basis of explicitly defined schema information and well-defined semantic vocabularies the choice of which depends on the concrete realization of a context provider $cp \in CP$.

As outlined, context providers specify the data they require and provide in form of a data description (see Section 4.4.1) that serves as a basis for routing context models $m \in M$ between compatible context providers $cp \in CP$. A data description allows for the computation of relations existing between compatible context providers $cp \in CP$ in terms of the contextual data they both acquire and require for performing their acquisition tasks. These relations are analyzed and computed by the context framework and recorded in the *compatibility matrix* $\mathbf{M}^C$ (cf. Section 4.4.2.4). Further details about this process are given in Section 4.4. In the following, we provide a definition of the data description being referred to in the context provider's definitions:

> **Definition 4.5** (Data Description)**.** Let *ddesc* denote a data description of a context provider $cp \in CP$ that formally specifies the vocabulary elements a context provider makes use of for describing the contextual information represented in the context model $m \in \mathbb{M}$ it emits and the elements an input context model m_{input} must consist of so that it can be processed by a complementary context provider $c \in C$. A data description is defined on the basis of a specific *data description ontology* (see Section 4.4.1) and serves as a basis for determining the specific type of a context provider, i.e., whether a context provider is a member of P or C.

The explicit specification of the vocabularies and the vocabulary elements a context provider $cp \in CP$ makes use of for describing the context models it emits and requires allows for computing

[15] A context provider is free to define its internal processing logic individually; this aspect is reflected in the formal model.

[16] We added the symbol of the context provider a data description *ddesc* pertains to as subscript to the power set to indicate this correspondence.

compatibility scores in terms of the vocabulary elements used for the representation of contextual information that enables a data-dependent orchestration of context providers $cp \in CP$ for context augmentation, refinement, and complementation. Therefore, the context framework analyzes the data descriptions exposed by context providers and dynamically routes data between compatible context providers based on the type of context information they provide and require. Detailed information about this process as well as the structure of the data description ontology and the elements it consists of will be given in Section 4.4.

To facilitate this kind of cooperation, context providers are orchestrated in so-called orchestration trees (see Section 4.3.4) that form directed acyclic graphs consisting of compatible context providers $cp \in V_p$. Within such a graph, a primary context provider $p \in P$ represents the *starting node* while complementary context providers $c \in V_p$ represent *adjacent nodes*. Edges represent data flows between context providers, i.e., they indicate compatibility in terms of contextual data so that the data emitted by one context provider can be further processed by adjacent context providers. This aspect is described in detail in the next section.

4.3.4 Orchestration Trees

The framework cascades context providers within so-called *orchestration trees* that resemble concepts of a workflow scheme. They allow for a controlled execution of combined context acquisition workflows in a decoupled manner while maintaining data and process consistency as well as data accurateness and data completeness with respect to the context providers $cp \in CP$ currently deployed within an instance of the framework. This form of context provider orchestration allows for an efficient acquisition and aggregation of contextual information while taking into account mobile operating system peculiarities (see [FZ94, KLO+04]).

The compatibility scores computed from the context providers and their data descriptions are recorded in a compatibility matrix $\mathbf{M}^C$ that serves as a basis for cascading compatible context providers. Compatible context providers are able to mutually process the context models $m \in M$ of other context providers as there exists a compatibility in terms of the vocabularies they make use of for describing the data they acquire and emit as well as the data they require and complement in the course of performing their acquisition tasks. Those compatibilities are reflected in orchestration trees that resemble a formal specification of the acquisition workflow schemes between compatible context providers depending on the context model elements specified in their data descriptions. For instance, if a context provider cp_1 is compatible to a context provider cp_2 and cp_2 itself is compatible to the context providers cp_3 and cp_4 such that the context model $m \in M$ emitted by the previously mentioned context provider is processed by the subsequent context providers by means of m^{input} (cf. Definition 4.4), they are orchestrated in an orchestration tree $o_p \in O$ such that cp_1 is precedent to cp_2 and cp_2, in turn, is predecessor of both cp_3 and cp_4. In essence, an orchestration tree is a well-structured specification of a data-dependent context acquisition workflow scheme between, in terms of their context models, compatible context providers where the degree of compatibility is recorded as a numerical score in the compatibility matrix $\mathbf{M}^C$ being the result of an analysis of the context providers' data descriptions. A formal and algorithmic description of the orchestration process is given in Section 4.4.

For defining an orchestration tree $o \in O$, we make use of elements from graph theory and represent an orchestration tree $o \in O$ as a *directed acyclic graph (DAG)* defined as the quadruple $G = (V, E, \alpha, \omega)$ where V represent the graph's vertices and corresponds to the context providers $cp \in CP$ being orchestrated within an orchestration tree $o \in O$. The set E represents the edges of a graph and corresponds to the relations existing between compatible context providers as

recorded in the orchestration matrix $\mathbf{M}^O$. α and ω are projections and specify the precedent vertex ($\alpha(e)$) as well as the succeeding or adjacent vertex ($\omega(e)$) of a concrete edge $e \in E$ existing between two elements $v_i, v_j \in V$ where $v_i \neq v_j$. Therefore, $\alpha(e)$ represents the vertex $v_i \in V$ corresponding to the *tail* of an edge $e \in E$ and $\omega(e)$ the vertex v_j connected with the *head* of the edge $e \in E$ for $v_i \neq v_j$.

For each primary context provider $p \in P$, we can derive an orchestration tree $o_p \in O$ from the orchestration matrix $\mathbf{M}^O$ (cf. Figure 4.8 on page 136) whose root element is always a primary context provider $p \in P$ and the adjacent nodes represent complementary context providers $c \in C$ that complement the data acquired by the corresponding primary context provider $p \in P$.

As a consequence, there exists per definition always a 1:1 relationship between a primary context provider $p \in P$ and its corresponding orchestration tree $o_p \in O$ where this relationship is expressed through the bijective relation $R_{corresponds}$ (cf. Table 4.2) existing for all $p \in P$ and $o \in O$ where $|O| = |P|$. Hence, for every $p \in P$ there exists a direct equivalent $o \in O$ which is denoted as o_p to indicate the 1:1 correspondence between elements $o \in O$ and $p \in P$ such that

$$\forall p \in P \, \exists! \, o \in O : (p, o_p) \in R_{corresponds} \tag{4.9}$$

By using the formal notation of directed acyclic graphs as a basis for formally defining the structural orchestration of compatible context providers in terms of the context-relevant data they require and emit, and by considering the fact that there always exists a 1:1-relationship between a primary context provider $p \in P$ and an orchestration tree $o_p \in O$ such that $p \sim_{R_{corresponds}} o_p$ where $|R_{corresponds}| = |O| = |P|$, we formally define an orchestration tree $o_p \in O$ as follows:

Definition 4.6 (Orchestration Tree)**.** An orchestration tree $o_p \in O$ for a specific primary context provider $p \in P$ is defined as the quadruple $o_p = (V_p, R_p, \alpha, \omega)$ with the following properties:

1. V_p is defined as the finite and non-empty set of the primary context provider p and its compatible adjacent complementary context providers $c \in C$ deduced from the orchestration matrix $\mathbf{M}^O$ and the compatibility matrix $\mathbf{M}^C$.
2. R_p is defined as the set of relations r that exist between the context provider p and compatible complementary context provider $c \in V_p$ where $r \in R_p$ is defined as the tuple
$$r \in \Big(\{p \in V_p\} \times \big(V_p \backslash \{p\}\big)\Big) \cup \Big(V_p \backslash \{p\}\Big)^2 \tag{4.10}$$
3. $\alpha : R_p \to V_p$ and $\omega : R_p \to V_p$ are projections where $\alpha(r)$ is the starting or direct *preceding* context provider and $\omega(r)$ the ending or direct *succeeding* context provider[17]of the relation $r \in R_p$. $\alpha(r)$ and $\omega(r)$ are adjacent to each other for a relation $r \in R_p$ while being incident to the relation $r \in R_p$.

The definition of the two projections $\alpha : R_p \to V_p$ and $\omega : R_p \to V_p$ allows us to define the set of context providers $p, c \in V_p$ orchestrated within an orchestration tree $o_p \in O$ more precisely; we therefore introduce the finite set $C_p \subset V_p$ that denotes the set of complementary context providers $c \in C$ that are orchestrated in the orchestration tree $o_p \in O$ of the corresponding primary context provider $p \in P$. On a formal level, we define C_p on the basis of the incident

[17]As per Definition 4.4, a succeeding, i.e., a context provider $cp \in V_p$ that is positively incident to a relation $r \in R_p$ such that $\omega(r) = cp$ is always a complementary context provider.

relations $r \in R_p$ in which a complementary context provider $c_i \in C$ is adjacent to a preceding context provider $c_j \in C$ for $i \neq j$ such that:

$$C_p := \big\{\omega(r) : r \in R_p\big\} \quad \Longrightarrow \quad C_p = \bigcup_{r \in R_p} \omega(r) \tag{4.11}$$

As a consequence, Equation (4.11) allows us to formally define V_p as the union of the primary context provider $p \in P$ and the complementary context providers $c \in C_p$:

$$V_p := \big\{p \in P : \alpha(r) = p\big\} \cup C_p \quad \text{where} \;\; r \in R_p \tag{4.12}$$

In addition, from the postulation that a context provider $cp \in CP$ can only be part of one specific orchestration tree $o_p \in O$, we can deduce that any two sets V_{p_i} and V_{p_j} where $i \neq j$ are disjunct with $V_{p_i} \cap V_{p_j} = \emptyset$. As a logical consequence, the union of all sets V_p is a subset of the set CP:

$$\bigcup_{i=1}^{|P|} V_{p_i} \subseteq CP \tag{4.13}$$

Based on the equations (4.12) and (4.13), any two sets C_{p_i} and C_{p_j} where $i \neq j$ are also disjunct and thus $C_{p_i} \cap C_{p_j} = \emptyset$. Consequently, as a set C_p is a real subset of V_p with $C_p \subset V_p$, the union over all sets C_p is a subset of the set C:

$$\bigcup_{i=1}^{|P|} C_{p_i} \subseteq C \tag{4.14}$$

A corollary of the disjointedness of any sets C_p for all $p \in P$ and Equation (4.14) is that any two sets V_{p_i} and V_{p_j} for $i \neq j$ are consequently also pairwise disjunct; this aspect is formulated in Equation (4.15):

$$\bigcap_{p \in P} V_p = \emptyset \tag{4.15}$$

After having defined the set V_p more precisely, we now define a number of additional sets and functions that are necessary for the formulation of the orchestration rules, i.e., the orchestration logic that serve as a basis for building the orchestration trees $o_p \in O$. For the following definitions, let $v \in V_p$ represent an arbitrary element referring to either a primary context provider $p \in V_p$ or a complementary context provider $c \in V_p$. With this convention, let $\delta^+_{o_p}(v)$ return the set of relations $r \in R_p$ to which a context provider $v \in V_p$ is positively incident such that $\alpha(r) = v$. Hence, $\delta^+_{o_p}(v_i)$ returns all outbound relations $r \in R_p$ that are incident to a context provider $v_i \in V_p$ such that $\exists\, r \in R_p : \alpha(r) = v_i \wedge \omega(r) = v_j$ for $i \neq j$ where $v_j \in V_p$ is a compatible complementary context provider being adjacent to v_i; thus, we can define $\delta^+_{o_p}(v) : V_p \longrightarrow R_p$ as follows:

$$\delta^+_{o_p}(v) := \big\{r \in R_p : \alpha(r) = v\big\} \tag{4.16}$$

Furthermore, let $\delta^-_{o_p}(v)$ return the set of relations $r \in R_p$ to which a context provider $v \in V_p$ is incident to such that $\omega(r) = v$; hence, $\delta^-_{o_p}(v_i)$ refers to the inbound relations to which a context provider $v_i \in V_p$ is adjacent to a preceding context provider $v_j \in V_p$ such that $\exists\, r \in R_p : \alpha(r) = v_j \wedge \omega(r) = v_i$ for $i \neq j$. Consequently, $\delta^-_{o_p}(v) : V_p \longrightarrow R_p$ is defined as:

$$\delta^-_{o_p}(v) := \big\{r \in R_p : \omega(r) = v\big\} \tag{4.17}$$

Generally, the sum of the results of both functions $\delta^+_{o_p}(v)$ and $\delta^-_{o_p}(v)$ is also denoted as the *valency* or *degree* of a vertex $v \in V$ pertaining to a graph $G = (V, E)$.

Moreover, let $N^+_{o_p}(v) : R_p \longrightarrow V_p$ denote the set of neighbors to $v \in V_p$ that are incident to the relations $r \in \delta^+_{o_p}(v)$ such that $\alpha(r) = v$ wherefore the result of $N^+_{o_p}(v_i)$ is the set of complementary context providers $v_j \in V_p$ that are adjacent to the context provider $v_i \in V_p$ for $i \neq j$. Thus, we define the set $N^+_{o_p}(v)$ as follows:

$$N^+_{o_p}(v) := \left\{\omega(r) : r \in \delta^+_{o_p}(v)\right\} \tag{4.18}$$

In the same way, let $N^-_{o_p}(v_i)$ denote the set of context provider $v_j \in V_p$ that are incident to a relation $r \in \delta^-_{o_p}(v_i)$ such that $\omega(r) = v_i$; therefore $N^-_{o_p}(v_i)$ represents the set of context providers $v_j \in V_p$ to which a context provider $v_i \in V_p$ is adjacent to for $i \neq j$. As a consequence, we define the set $N^-_{o_p}(v)$ as follows:

$$N^-_{o_p}(v) := \left\{\alpha(r) : r \in \delta^-_{o_p}(v)\right\} \tag{4.19}$$

In consequence, both sets $N^+_{o_p} : R_p \longrightarrow V_p$ and $N^-_{o_p} : R_p \longrightarrow V_p$ are defined as the results of a projection existing between the sets R_p and V_p of an orchestration tree $o_p \in O$.

4.3.5 Formal Definition of the Orchestration Logic

Based on the previous definitions about orchestration trees, we now formally define the orchestration logic underlying an orchestration tree $o_p \in O$ in an axiomatic way. These axioms or orchestration rules serve as a basis for building the orchestration trees $o_p \in O$ deduced from the orchestration matrix $\mathbf{M}^O$ and are mandatory for the correct interpretation and processing of the context acquisition workflows constituted by the containing context providers $p, c \in V_p$:

1. As an orchestration tree $o_p \in O$ per definition always has a primary context provider $p \in V_p$ as its root respectively starting element, such primary context provider $p \in V_p$ is not adjacent to any other context provider such that $\omega(r) = p$. This aspect is formulated in Equation (4.20):

   $$\forall p \in V_p \wedge p \in P \nexists r \in R_p : \omega(r) = p \tag{4.20}$$

   Consequently, the set of relations $r \in R_p$ a primary context provider $p \in V_p$ is incident to such that $\omega(r) = p$ is empty. Thus, also the set of its neighbors $N^-_{o_p}(p)$ to which $p \in V_p$ is adjacent to is empty:

   $$\delta^-_{o_p}(p) = \emptyset \implies N^-_{o_p}(p) = \emptyset \quad \text{for all } p \in V_p \tag{4.21}$$

2. Due to the fact that any two sets V_{p_i} and V_{p_j} are pairwise disjunct for all $p \in P$ and $i \neq j$ (see Equation (4.15)), any complementary context provider $c_i \in V_p$ can as a corollary only be adjacent to one specific context provider $c_j \in V_p$ for $i \neq j$ such that $\exists! \, r \in R_p : \omega(r) = c_i$. As of Equation (4.17) and (4.19), the sets $\delta^-_{o_p}(c)$ and $N^-_{o_p}(c)$ for all complementary context providers $c \in C_p$ are thus:

   $$\left|\delta^-_{o_p}(c)\right| = \left|N^-_{o_p}(c)\right| = 1 \tag{4.22}$$

   Since an orchestration tree $o_p \in O$ is defined as a directed acyclic graph $o_p = (V_p, R_p, \alpha, \omega)$ (cf. Definition 4.6) where each complementary context provider $c \in C_p$ can only be adjacent to one specific context provider, we can deduce that a complementary context provider

$c \in V_p$ can never have two or more preceding context providers $v_i, v_j \in V_p$ where $c \neq v_i, v_j$ and $i \neq j$:

$$\forall c \in C_p \, \exists! \, r_i \in R_p : \omega(r_i) = c \quad \text{for} \quad 1 \leq i \leq |R| \tag{4.23}$$

3. Based on Equation (4.23) and due to the fact that an orchestration tree $o_p \in O$ is defined as a directed acyclic graph $o_p = (V_p, R_p, \alpha, \omega)$, a context provider $cp \in V_p$[18] can never be adjacent to itself, that is, be both predecessor and successor of one particular relation $r \in R_p$. A context provider $p, c \in V_p$ can per definition not consume the context model m it produces; therefore, we define the following convention:

$$\forall r \in R_p : \alpha(r) \neq \omega(r) \tag{4.24}$$

4. In addition to the equations (4.15), (4.23), and (4.24) and due to the fact that an orchestration tree $o_p \in O$ is defined on the basis of a directed acyclic graph, there does not exist a cyclic path between the context providers $p, c \in V_p$: Therefore, let ψ_p denote a path of an orchestration tree $o_p \in O$ and defined as follows:

$$\psi_p \in V_p \times \big(V_p \setminus p\big)^{|R_p - 1|} \tag{4.25}$$

Moreover, let $\psi[i]$ be a lookup function that returns the context provider located at the i-th position in ψ_p where $1 \leq i \leq |R_p|$. The lookup function together with Equation (4.25) allows us to define the non-cyclic requirement of orchestration trees $o_p \in O$ on a formal basis:

$$\forall \, \psi_p \in \left(V_p \times \big(V_p \setminus p\big)^{|R_p - 1|} \right) \nexists \, \psi[i], \psi[j] : \psi[i] = \psi[j] \quad \text{for } i \neq j \tag{4.26}$$

5. There might be situations in which there exists no complementary context providers $c \in C$ that refine the data emitted by a primary context provider $p \in V_p$ in a running instance of the replication framework where $C_p = \emptyset$ and $|V_p| = 1$, that is, V_p consists of only one primary context provider $p \in P, V_p$. We treat such a degenerated orchestration tree that consist of only a starting or root node $p \in V_p$ while exhibiting no other relations $r \in R_p$ to any complementary context providers as a special case in which

$$\delta^+_{o_p}(p) \, = \, \delta^-_{o_p}(p) \, = \, N^+_{o_p}(p) \, = \, N^-_{o_p}(p) \, = \, \emptyset. \tag{4.27}$$

Although such an orchestration tree $o_{p_i} \in O$ represents a special case, it is treated equally compared to orchestration trees $o_{p_j} \in O$ in which $C_{p_j} \neq \emptyset$ for $p_i \neq p_j$.

With Definition 4.6 and the set of rules constituting the underlying orchestration logic, an orchestration tree $o_p \in O$ can be expressed by means of an *adjacency tableau*, as exemplarily depicted in Figure 4.2, that represents the relations $r \in R_p$ that exist between the adjacent context providers $p, c \in V_p$ being orchestrated in the orchestration tree $o_p \in O$ together with $\alpha(r_i \in R_p)$ and $\omega(r_i \in R_p)$ iff $C_p \neq \emptyset$. In addition, the definition of the two functions $\alpha : R_p \longrightarrow V_p$ and $\omega : R_p \longrightarrow V_p$ allows for expressing a relation $r_i \in R_p$ as $r_i = \big(\alpha(r_i), \omega(r_i)\big)$ that serves as a basis for building the adjacency tableau for an orchestration tree $o_p \in O$ being created for a primary context provider $p \in P$.

The adjacency tableau depicted in Figure 4.2 shows the relations $r \in R_{p_1}$ for an exemplary orchestration tree $o_{p_1} \in O$ that exist between the context providers $p_1, c_{1,2,4,5,8} \in V_{p_1}$ orchestrated

[18] Please note that we use the symbol *cp* in this definition as this requirement applies to both a primary context provider $p \in V_p$ as well as a complementary context provider $c \in V_p$.

	α	ω
r_1	p_1	c_1
r_2	p_1	c_2
r_3	c_2	c_4
r_4	c_2	c_5
r_5	c_5	c_8

FIGURE 4.2: Example of an adjacency tableau created from the elements $r \in R_p$ for a context provider $p \in P$

within this exemplary orchestration tree. In this example, the set V_{p_1} consists of six context providers with $V_{p_1} = \{p_1, c_1, c_2, c_4, c_5, c_8\}$ and the following concrete relations $r \in R_{p_1}$ that exist between them: $R_{p_1} = \big\{r_1 = (p_1, c_1), r_2 = (p_1, c_2), r_3 = (c_2, c_4), r_4 = (c_2, c_5), r_5 = (c_5, c_8)\big\}$.

For instance, the function $\alpha(r_3)$ for the relation $r_3 \in R_{p_1}$ returns the preceding context provider $\alpha(r_3) = c_2$, and the function $\omega(r_3) = c_4$ the succeeding context provider $c_4 \in V_{p_1}$ of the relation $r_3 \in R_{p_1}$, i.e., those context providers that are incident to the relation $r_3 \in R_{p_1}$. This relation indicates that there exists a compatibility in terms of context models between these two context providers where the context model $m \in M, M_{c_2}$ created by the preceding context provider $c_2 \in V_{p_1}$ is consumed by a succeeding context provider $c_4 \in V_{p_1}$. In the same way, the function $\delta^+_{o_p}(c_2)$ returns all outbound relations $r \in R_{p_1}$ to which c_2 is incident to such that $\alpha(r) = c_2$. For c_2, these relations are $\delta^+_{o_p}(c_2) = \{r_3, r_4\}$. Likewise, the set of neighbors $N^+_{o_p}(c_2)$ being adjacent to the given context provider $c_2 \in V_p$ is defined by Equation (4.18) and therefore $N^+_{o_p}(c_2) = \{c_4, c_5\}$. In addition, the set of incident relations and precedent context providers for the given context provider c_2 from the present example are $\delta^-_{o_p}(c_2) = \{r_2\}$ and $N^-_{o_p}(c_2) = \{p_1\}$, which is the primary context provider $p_1 \in P$ for the orchestration tree o_{p_1}.

The adjacency tableau (cf. Figure 4.2) used to describe the structure of an orchestration tree $o_{p_1} \in O$ can also be transformed into an $|V_p| \times |V_p|$ adjacency matrix $\mathbf{M}^A$ which we denote as $\mathbf{M}^A_{o_p}$ for a primary context provider $p \in P$ where $\mathbf{M}^A_{o_p}\big(\alpha(r_i), \omega(r_i)\big) = 1$ in case there exists a relation $r_i \in R_p$ between two context providers $\alpha(r_i), \omega(r_i) \in V_p$ where $\alpha(r_i) \neq \omega(r_i)$. A matrix coefficient is thus defined as follows:

$$\mathbf{M}^A_{o_p}\big(\alpha(r_i), \omega(r_i)\big) = \begin{cases} 1 & \iff r_i \in R_p \\ 0 & otherwise \end{cases} \tag{4.28}$$

4.3.6 Compounded Context Acquisition Model

An orchestration tree $o_p \in O$ for a primary context provider $p \in P$ is considered as an atomic unit (cf. Section 4.5), which, when all acquisition tasks of the contained context providers $p, c \in V_p$ have finished, provides a compound context acquisition model denoted as M_{o_p} that is composed of and aggregated from the most recently emitted context models $m \in M$ of the context providers $p, c \in V_p$ orchestrated in the orchestration tree $o_p \in O$. Since $o_p \in O$ denotes the orchestration tree corresponding to a primary context provider $p \in P$, let M_{o_p} therefore denote the compounded context model aggregated from the context models $m \in M$ acquired by the context providers $p, c \in V_p$. Based on the Equations (4.4) and (4.5), we define the aggregated context acquisition model $M_{o_p} \in OM$ corresponding to an orchestration tree $o_p \in O$ as the union of the most recently emitted context models m indicated by the function $\phi_{current} : M \longrightarrow M$ applied to the set of emitted context models M_p and M_c of the containing context providers $p, c \in V_p$.

Definition 4.7 (Compounded Context Acquisition Model)**.** Let M_o denote a general compound context acquisition model and let $M_{o_p} \in OM$ denote a compounded context acquisition model corresponding to the orchestration tree $o_p \in O$ that is composed of the most recently emitted context models $m \in M$ created by the context providers $p, c \in V_p$ orchestrated in o_p. It is created when all context providers $p, c \in V_p$ have emitted their context models $m \in M_p$ or $m \in M_c$ for all $c \in C_p$. The set M_{o_p} is therefore defined as the union of the currently emitted context model $m \in \phi_{current}(M_p)$ and the currently emitted context models $m \in \phi_{current}(M_c) : \forall c \in C_p$ and formally expressed in Equation (4.29):

$$M_{o_p} := \left\{\phi_{current}(M_c) : c \in C_p\right\} \cup \phi_{current}(M_p) \tag{4.29}$$

Furthermore, let OM denote the set of all compounded context acquisition models M_{o_p}. As there exists a 1:1 relationship between an orchestration tree $o_p \in O$ and its corresponding compound context acquisition model $M_{o_p} \in OM$ aggregated from the single context models $m \in M_{o_p}$, we define the binary bijective relationship $R_{acquires} : O \times OM$ (cf. Table 4.2) between them as follows:

$$\forall o_p \in O \, \exists! \, M_{o_p} \in OM : (o_p, M_{o_p}) \in R_{acquires} \tag{4.30}$$

Based on Equation (4.4) and (4.5) and due to the fact that the sets M_p and M_c, for all $p \in P$ and $c \in C$, are partitions of M and their union is thus a subset of M (see Equation (4.3)) we can as a corollary state that the union of all sets $M_{o_p} \in OM$ must also be a subset of M:

$$\bigcup_{p \in P} M_{o_p} \subseteq M \tag{4.31}$$

Note that any two sets $M_{o_{p_i}}, M_{o_{p_j}} \in OM$ for $i \neq j$ do not necessarily have to be disjunct such that $\bigcap_{i=1}^{|O|} M_{o_{p_i}} \neq \emptyset$ as the instances of two different context providers $cp_i, cp_j \in CP$ for $i \neq j$ can deliver an identical context model instance $m \in \mathbb{M}$ although orchestrated in different orchestration trees $o_{p_i}, o_{p_j} \in O$. However, it is the context dispatcher's task (cf. Figure 4.10 on page 158) to recognize and merge such models.

4.3.7 Context Acquisition Workflow

As discussed in Section 4.3.4, orchestration trees $o_p \in O$ are autonomous, self-contained, and dynamically created representations of context acquisition workflow schemes that specify the composition of compatible context providers $p, c \in V_p$ into logical units for context refinement, complementation, and augmentation. This requires that an orchestration tree $o_p \in O$ needs to be deployed in an execution environment[19] that is able to control and monitor the acquisition processes of the orchestrated context providers $p, c \in V_p$ on the basis of a specific and predefined processing logic (which will be discussed in Section 4.5). This task is carried out by context acquisition workflows, denoted as wf (cf. Table 4.1), which represent a controlled and autonomously operating environment for the execution of the elementary context acquisition workflows of the context providers $p, c \in V_p$ orchestrated in an orchestration tree $o_p \in O$. Therefore, we define a context acquisition workflow $wf \in WF$ as follows:

[19] Such execution environments are called *sandbox* in other works and related domains.

Definition 4.8 (Context Acquisition Workflow)**.** Let wf denote a context acquisition workflow and let WF denote the set of all context acquisition workflows $wf \in WF$ currently existing in a running instance of the proposed framework. Let us further denote the context acquisition workflow for an orchestration tree $o_p \in O$ as $wf_{o_p} \in WF$ to indicate this correspondence. Since there exists a binary 1:1 relationship between an orchestration tree $o_p \in O$ and its corresponding context acquisition workflow $wf_{o_p} \in WF$, let the relation $R_{executedBy} : O \times WF$ (cf. Table 4.2) define such a relation. The relation $R_{executedBy}$ is bijective as there exists for any given point in time t_n exactly one relation $r_i \in R_{executedBy}$ between a concrete context acquisition workflow $wf_{o_p} \in WF$ and the corresponding orchestration tree $o_p \in O$ it offers its execution environment to. This fact is postulated in Equation (4.32):

$$\forall o_p \in O \; \exists! \, wf_{o_p} \in WF : (o_p, wf_{o_p}) \in R_{executedBy} \tag{4.32}$$

This formal definition admittedly is rather general as a context acquisition workflow $wf \in WF$ is principally an implementation-specific construct to monitor and control the execution of the elementary context acquisition tasks of the containing context providers $p, c \in V_p$ of the corresponding orchestration tree $o_p \in O$. A context acquisition workflow $wf_{o_p} \in WF$ becomes active whenever the primary context provider $p \in P$ of its associated orchestration tree $o_p \in O$ detects a change in the contextual data it acquires and, as a consequence of that change, emits an new context model $m \in M, M_p$. Since an orchestration tree $o_p \in O$ is a tree-based structured specification of a context acquisition workflow scheme, a context acquisition workflow $wf \in WF$ can be regarded as an interpreter and compiler of such a workflow specification as it is able to initiate and monitor the respective context acquisition tasks while ensuring that parallel running context provider acquisition workflows do not interfere and that the context framework remains in a deterministic and controlled state. Further information about context acquisition workflows are given in Section 4.5 including an algorithmic specification of the processing logic underlying a context acquisition workflow.

4.3.8 Context Configuration

When the context models $m \in \mathbb{M}$ of all context providers $cp \in V_p$ being part of all orchestration trees $o_p \in O$ were acquired successfully, a global context model, the so-called *context configuration* is created. The context configuration is an aggregated and consolidated representation of all acquired context models $m \in \mathbb{M}$ contained in the compound context acquisition models $M_{o_p} \in OM$ that correspond to the orchestration trees $o_p \in O$ and serves as mean to built a global and coherent model of the user's current context. In Definition 4.9, we define a context configuration on a formal basis:

Definition 4.9 (Context Configuration)**.** Let cc be an instance of a context configuration and CC be the set of all context configuration instances $cc \in CC$ created by a running instance of the replication framework. An instance of a context configuration $cc \in CC$ is the result of a 2-parametric function $f_{consolidate} : \mathbb{M}^k \times \Pi^l \longrightarrow CC$[20]that takes an arbitrary but finite amount of context models $m \in \mathbb{M}$ and applies a finite number of reasoning rules $\pi \in \Pi$, where k represents the amount of context models pertaining to the union of all compound context acquisition models $M_{o_p} \in OM$ and l represents a finite number of reasoning rules $\pi \in \Pi$:

$$f_{consolidate} : \mathbb{M}^k \times \Pi^l \longrightarrow CC \quad \text{where} \quad k = \left| \bigcup_{p \in P} M_{o_p} \right|, \; l \leq |\Pi| \tag{4.33}$$

The function $f_{consolidate}$ processes the context models m contained in the compound context acquisition models $M_{o_p} \in OM$ of all orchestration trees $o_p \in O$ and infers additional information or consolidates existing information by applying reasoning techniques and a distinct amount of reasoning rules $\pi \in \Pi$. The application of reasoning rules $\pi \in \Pi$ to the context models m contained in the compound context acquisition models $M_{o_p} \in OM$ cause the creation of an updated, i.e., consolidated set of compound context acquisition models that incorporates a distinct number of inferred statements in addition to the set of asserted statements explicitly represented by the context models $m \in M_{o_p}$. Moreover, a reasoning rule $\pi \in \Pi$ might also cause the complete merging of two autonomously acquired context model as illustrated in Figure 5.7 on page 183, where location-based information is merged into a common sub graph. As a consequence, an instance of a context configuration $cc \in CC$ is the result of the function $f_{consolidate} : \mathbb{M}^k \times \Pi^l \longrightarrow CC$ and hence also a member of $\mathbb{M}$.

Furthermore, we likewise define two functions $\phi_{current} : CC \longrightarrow CC$ and $\phi_{previous} : CC \longrightarrow CC$ for the set of context configuration instances CC, where $\phi_{current}(CC)$ returns the most recently created context configuration instance $cc_{\tau_{i+1}} \in CC$ and $\phi_{previous}(CC)$ the previously created context configuration instance $cc_{\tau_i} \in CC$ by the function $f_{consolidate} : \mathbb{M}^k \times \Pi^l \longrightarrow CC$. This allows context consumers to reason about changes respectively transitions in the global representation of the user's context between two different points in time τ_i and τ_{i+1} where $\phi_{previous}(CC)$ refers to a context configuration instance valid at time τ_i and $\phi_{current}(CC)$ to an instance valid at time τ_{i+1}[21].

4.3.9 Context Description

Context providers $cp \in CP$ exchange the context models $m \in \mathbb{M}$ they create and emit through a context description $d \in D$ that not only contains the context model $m \in \mathbb{M}$ as such but also additional information necessary for its processing and aggregation. A context description thus serves as a "transportation vehicle" for communicating contextual information represented in context models $m \in \mathbb{M}$ where there exists a 1:1 relation between an emitted context model $m \in M$ and the context description $d \in D$ it is wrapped by. Let the relation $R_{convey} : D \times M$ therefore denote the relationship existing between an emitted context model $m \in M$ and the

[20]The function $f_{consolidate} : \mathbb{M}^k \times \Pi^l \longrightarrow CC$ in this context is defined to perform both aggregation and consolidation heuristics.

[21]We assume that τ_i is represented as an absolute value (e.g., in milliseconds) calculated from a specific and fixed point τ_0 in time where $\tau_0 \leq \tau_i \leq \tau_{i+1}$.

context description $d \in D$ conveying m. In the following, we formally define a context description d and the set of all context descriptions D:

> **Definition 4.10** (Context Description)**.** A context description $d \in D$ is defined as a 5-tuple $(cp, m, \rho, \sigma, \tau)$ where $m \in \mathbb{M}$ represents the context model emitted by the context provider $cp \in CP$; ρ represents status information and σ a status code that allows for adjusting processing and aggregation tasks based on the status of the corresponding context provider $cp \in CP$ at the time t_n when the context model $m \in \mathbb{M}$ was created. The symbol τ denotes the time stamp when the context model $m \in \mathbb{M}$ was emitted for facilitating synchronization purposes.

The distinction between a context description $d \in D$ and a context model $m \in \mathbb{M}$ is important as it allows for the specification of additional information associated with the acquisition process in which a context model $m \in \mathbb{M}$ was created and to deal with the diversity and heterogenous nature of context sources as such. Several works (e.g. [BS03, KMK+03, BTC06, PvHS07, SWvS07, MTD08]) suggest to include a mechanism related to the determination of the *quality* and *trustworthiness* of acquired contextual information, which is particularly important for ubiquitous computing environments in which a multitude of different and heterogeneous context sources have to be exploited dynamically. The concept of *Quality of Context (QoS)* thus elaborates around one of the most prevalent problems of context-aware computing, that is, the circumstance that real-world situations can not be represented 100% accurate wherefore contextual information is inherently *vague*, *incomplete*, and *inaccurate* [PvHS07]. Context descriptions in particular are therefore means to specify such Quality of Context indicators (e.g., a selection of indicators such as *precision*, *freshness*, *temporal resolution*, *spatial resolution*, and *probability of correctness* has been specified in [PvHS07]) that may not directly refer to the contextual information represented in a context model $m \in \mathbb{M}$ but facilitates subsequent processing and aggregation tasks as they allow for specifying meta information related to the information represented by a context model $m \in \mathbb{M}$ itself as well as to the context source $s \in S$ the context information originated from. By analyzing such data, a context consumer can elaborate on the trustworthiness and quality of acquired contextual information and adjust its processing tasks accordingly.

4.3.10 Data Providers

When a new instance of a context configuration $cc \in CC$ is built from the compound context acquisition models $M_{o_p} \in OM$ using the aggregation and consolidation function $f_{merge} : \mathbb{M} \times \Pi \longrightarrow CC$, it is forwarded to data providers $dp \in DP$ that analyze the global representation of the user's current context represented by a context configuration $cc \in CC$ and adapt and initiate their data replication tasks accordingly. In accordance to the definition of context providers (cf. Definition 4.3) we also make use of concepts from automata theory for defining data providers. As data providers replicate data to the mobile device as a result of an analysis of the current context configuration instance $cc \in CC$ representing the user's current context, they can be regarded as transducers and hence are defined as *minimal deterministic finite state transducers (DFST)*:

Definition 4.11 (Data Provider)**.** A data provider $dp \in DP$ is a minimal deterministic finite state transducer defined by the 6-tuple $(Q, E, m^{input}, m^{replica}, \delta, q_0)$ that allows for ε-transitions and consists of the following elements:

- Q is a set of states $q \in Q$ into which a data provider $dp \in DP$ transitions to in the course of replicating data to a mobile device.
- E is defined as a set of events $e \in E$, which, when an event e occurs, causes a transition from a current sate $q_i \in Q$ into a subsequent state $q_{i+1} \in Q$.
- $m^{input} \in \mathbb{M}$ denotes the context model that serves as input data for initiating a data replication task performed by a data provider $dp \in DP$. Usually, m^{input} corresponds to a context configuration $cc \in CC$.
- $m^{replica}$ represents the RDF data being replicated to the mobile device.
- $\delta : Q \times E \cup \{\varepsilon\} \longrightarrow Q$ is a transition function that, based on the occurrence of a specific event $e \in E$, defines a transition into a subsequently following state $q_{i+1} \in Q$.
- q_0 is the initial state of a data provider $dp \in DP$ before it initiates its data replication workflow.

We refer to $m^{input} \in \mathbb{M}$ in the previous definition instead of $cc \in CC$ as instances of context configurations exhibit the same characteristics as context models and as a corollary can be regarded as complex context models. We also included ε-transitions in the definition of the transition function $\delta : Q \times E \cup \{\varepsilon\} \longrightarrow Q$ as a transition from a state $q_i \in Q$ into $q_{i+1} \in Q$ might also be conducted independently from the occurrence of a specific event $e \in E$. In Section 4.6 and 5.2, we provide a more technically focused description of data providers.

After having defined the elements constituting the formal model on a theoretical basis, we now provide a formal definition of the building process underlying the orchestration trees $o_p \in O$.

4.4 Formal Model of the Orchestration Process

The idea of orchestrating context providers is to augment and complement contextual information to build richer, more expressive, and elaborated context descriptions while maintaining data and processing consistency as well as data accurateness and data completeness. The orchestration of context providers $cp \in CP$ enables an augmentation of the context-relevant information space acquired by a primary context provider $p \in P$ through a set of compatible complementary context provider $c \in V_p$ that take the context model $m \in M_p$ emitted by $p \in P$ as input data for initiating their context acquisition tasks in order to mutually complement the information represented in $m \in M_p$. Based on an analysis of the data description *ddesc* of a context provider $cp \in CP$, we calculate a *similarity indicator* expressed numerically in a *similarity score* that indicates the degree of compatibility existing between two context providers $cp_i, cp_j \in CP$ for $cp_i \neq cp_j$ based on the vocabulary elements they use for representing contextual data. The rationale behind the proposed orchestration approach is that the contextual data a context provider delivers can be explicitly represented using vocabularies and ontologies from the Semantic Web. We therefore assume the existence of appropriate vocabularies that can be used for describing the data a context provider emits (e.g. [CPFJ04, KHK+04, Pre04, WZGP04, MYS05]). For complex

contextual constellations, different RDF vocabularies can be combined. We further assume that the vocabulary elements a context provider uses to describe and represent its data can serve as indicators for the type of context data it emits. If, for instance, a context provider makes use of the FOAF vocabulary, we can infer that it delivers personal or person-related data representing the social relationships and concerns of the user (e.g., the attendees of a meeting collected from the user's calendar, the user's friends etc). In this section, we now formally define and describe the relevant steps that constitute the building process and the deduction of orchestration trees $o_p \in O$. Generally, the building process can be compartmentalized in nine steps, which constitute as follows:

1. Compartmentalization of the set of context providers $cp \in CP$ into two partitions P and C depending on the vocabulary terms specified in the input and output sections of their data descriptions (cf. Section 4.4.1).
2. Creation of *inverted term indices* I from the vocabulary elements (terms t) specified in the input and output sections of the data description of each context provider $cp \in CP$.
3. Creation of a *merged inverted term index* I^{merged} from the inverted term indices I for a pair of context providers $cp_i, cp_j \in CP$ for $i \neq j$ their compatibility score is to be computed.
4. Transformation of the inverted term indices I and merged inverted term index i^{merged} into inverted term index vectors $\vec{V}^{index}$ that serve as vector representations of the terms t defined in the inverted term indices.
5. Calculation of several similarity scores between pairs of context providers $cp_i, cp_j \in CP$ for $cp_i \neq cp_j$ from the inverted term index vectors of their data descriptions *ddesc*.
6. Aggregation of the similarity scores calculated from the inverted term indices vectors of the context providers $cp \in CP$ into a compatibility vector $\vec{V}_{Compatibility}$.
7. Calculation of a compatibility matrix $\mathbf{M}^C$ on the basis of an analysis of the compatibility vectors $\vec{V}_{Compatibility}$ for each pair of context providers $cp_i, cp_j \in CP$ for $i \neq j$.
8. Transformation of the compatibility matrix $\mathbf{M}^C$ into the orchestration matrix $\mathbf{M}^O$ recording the adjacent relations existing between compatibly context providers $cp_i, cp_j \in CP$ for $i \neq j$.
9. Deduction of the orchestration trees $o_p \in O$ from the orchestration matrix $\mathbf{M}^O$ for each primary context provider $p \in P$.

The compatibility score computation is not commutative wherefore $\big(|CP|^2 - |CP|\big) - \big(|P|^2 - |P|\big)$ combinations are necessary for computing the compatibility scores for $|CP|$ context providers. Step 1 to 3 is described in Section 4.4.2.1. Section 4.4.2.2 outlines the transformation into inverted term index vectors (step 4). The similarity scores calculation and aggregation into compatibility vectors (step 5 and 6) is treated in Section 4.4.2.3. Section 4.4.2.4 details the transformation of the compatibility matrix $\mathbf{M}^C$ into the orchestration matrix $\mathbf{M}^O$ (step 7 and 8) whereas Section 4.4.3 elaborates on the deduction of orchestration trees $o_p \in O$ together with the necessary algorithms (step 9).

Before we describe the previously mentioned steps in building an orchestration tree $o_p \in O$ on a formal and theoretic basis, we give detailed insights into the structure and constituting terms of the data description ontology in order to provide the reader with basic knowledge on which the orchestration steps are built upon.

Table 4.4: Core classes of the data description ontology

Local Name	**rdf:type**	**rdfs:SubClassOf**
`ddesc:ContextProvider`	`rdfs:Class`	-
`ddesc:Specification`	`rdfs:Class`	-
`ddesc:InputSpecification`	`rdfs:Container`	`ddesc:Specification`
`ddesc:OutputSpecification`	`rdfs:Class`	`ddesc:Specification`
`ddesc:VocabularySpecification`	`rdfs:Class`	`ddesc:Specification`
`ddesc:ConceptSpecification`	`rdfs:Container`	`ddesc:VocabularySpecification`
`ddesc:PropertySpecification`	`rdfs:Class`	`ddesc:VocabularySpecification`
`ddesc:Namespace`	`rdfs:Class`	-

4.4.1 Data Description Ontology

We have defined a lightweight data description ontology that consists of a set of minimal[22] vocabulary elements and a set of rules that define the structure of a data description and the scope of its elements. We first introduce the classes and properties being defined in the data description ontology before we provide insights on how these elements can be used to specify a context provider's data description. We complement this description by means of a concrete example of a data description defined for a complementary context provider $c \in C$ that extracts contact data from acquired calendar data.

Table 4.4 and Figure 4.4 list the core classes that constitute a data description together with information about their specific type, indicated by the `rdf:type` property, and the classes they are inherited from (`rdfs:subClassOf`) forming the structure of the data description schema. The `ddsec:ContextProvider` class is used to describe a specific context provider $cp \in CP$ being identified through its `UUID` so that it can be integrated in the orchestration process. The type of all specification classes defined in the data description ontology is represented by the `ddesc:Specification` class that serves as super class for the specification of the vocabularies used for the input and output context model, indicated by the `ddesc:InputSpecification` and `ddesc:OutputSpecification` classes, as well as for the concrete specification of the vocabulary elements, indicated by the `ddesc:VocabularySpecification` class. The `ddesc:InputSpecification` class refers to the data description subgraph that describes the vocabulary elements that the input context model has to be built upon so that it can be processed by a context provider, whereas the `ddesc:OutputSpecification` class allows for specifying the vocabulary elements the context model created by a context provider consists of. A vocabulary specification itself is super class of the `ddesc:ConceptSpecification` and `ddesc:PropertySpecification` classes that allow for specifying detailed vocabulary elements being relevant for the specification of a data description. The last element in Table 4.4 is the `ddesc:Namespace` class, which allows for defining the vocabularies' namespaces.

In addition, Table 4.5 lists the properties being defined in the course of the data description ontology together with their `rdfs:domain` and `rdfs:range` values that define how a property is related to the classes defined in Table 4.4. These properties are essential for the specification of a data description as they define the structure a data description has to comply with in order to integrate the context provider it describes in the orchestration process. The `ddesc:input`

[22] Minimal in this context refers to the minimal set of constituting elements and rules necessary for defining the underlying schema and semantics of the data description ontology so that a data-dependent orchestration of compatible context providers can be realized.

Table 4.5: Domain and range values of the data description ontology properties

Property	Domain	Range
ddesc:input	ddesc:ContextProvider	ddesc:InputSpecification
ddesc:output	ddesc:ContextProvider	ddesc:OutputSpecification
ddesc:vocabulary	ddesc:Specification	ddesc:VocabularySpecification
ddesc:namespace	ddesc:VocabularySpecification	ddesc:Namespace
ddesc:concepts	ddesc:VocabularySpecification	ddesc:ConceptSpecification
ddesc:properties	ddesc:VocabularySpecification	ddesc:PropertySpecification
ddesc:mandatory	ddesc:VocabularySpecification	ddesc:Namespace
ddesc:optional	ddesc:VocabularySpecification	ddesc:Namespace

and `ddesc:output` properties exist between resources that can be identified as members of the class `ddesc:ContextProvider` and resources that specify the elements of the input and output sections of a data description, indicated by the classes `ddesc:InputSpecification` and `ddesc:OutputSpecification` respectively. They allow to refer to a subgraph of the data description as being the input specification respectively output specification of a context provider. The `ddesc:vocabulary` property has been defined for both the `ddesc:InputSpecification` and `ddesc:OutputSpecification` classes and identifies the resources used as domain values (in the relevant statements) as members of the `ddesc:VocabularySpecification` class, which can be deduced via inferencing. The `ddesc:VocabularySpecification` class is the domain of the three properties `ddesc:namespace`, `ddesc:concepts`, and `ddesc:properties` that define the necessary vocabulary elements in detail. The classes these properties specify as `rdfs:range` values refer to their respective names. The `ddesc:mandatory` and `ddesc:optional` properties have been defined for both `ddesc:ConceptSpecification` and `ddesc:PropertySpecification` classes and allow for defining those vocabulary terms that are considered as mandatory or optional for the representation of a context model. The range of both properties is the `ddesc:Namespace` class as these properties' values reference concrete vocabulary terms.

After having defined the classes and properties constituting the data description ontology, we now describe how these elements can be used to constitute a data description of a context provider so that it can be integrated in the orchestration process. In general, a data description consists of two main parts: an *input description* indicated by the `ddesc:input` property, and an *output description* indicated by `ddesc:output` property. The former specifies the data a context provider needs for performing its acquisition tasks whereas the later defines the elements used to describe the data a context provider emits. Both input and output specification are defined on the basis of the same schema; they contain a vocabulary specification—indicated by the `ddesc:vocabulary` property—that specifies the most relevant elements of a vocabulary such as its namespace, concepts, and properties in more detail.

An input specification may contain multiple `ddesc:vocabulary` properties, covering the case that context providers may be capable of processing data described with different vocabularies. Multiple vocabulary properties are interpreted in terms of the orchestration logic as alternatives, that is, they are interpreted as being connected with a logical *or*. This allows to specify vocabularies a context provider is capable to process in separate `ddesc:InputSpecification` sub graphs. An output specification per definition defines only one vocabulary specification subgraph although it might be imaginable that a context provider offers the opportunity to configure the vocabulary and thus the terms used to render its context model.

```
1   DATA_DESCRIPTION          ::= SPECIFICATION ;
2
3   SPECIFICATION             ::= [ INPUT_SPECIFICATION ] | OUTPUT_SPECIFICATION ;
4
5   INPUT_SPECIFICATION       ::= VOCABULARY_SPECIFICATION ;
6   OUTPUT_SPECIFICATION      ::= VOCABULARY_SPECIFICATION ;
7
8   VOCABULARY_SPECIFICATION ::= NAMESPACE , CONCEPTS , PROPERTIES ;
9
10  CONCEPTS                  ::= MANDATORY , OPTIONAL ;
11  PROPERTIES                ::= MANDATORY , OPTIONAL ;
12
13  MANDATORY                 ::= { URI } ;
14  OPTIONAL                  ::= { URI } ;
15  NAMESPACE                 ::= { URI } ;
```

FIGURE 4.3: Structure of the data description ontology represented in EBNF

A vocabulary specification consists of three parts: the `ddesc:namespace` property, which holds the vocabulary's namespace that is used for an upper-level orchestration, and the `ddesc:concepts` and `ddesc:properties` statements, which specify mandatory and optional concepts and properties that the context provider processes. Mandatory elements (indicated by the `ddsec:mandatory`-property) are those that are inevitable for a complementary context provider to perform its acquisition tasks. Optional elements (indicated by the `ddsec:optional`-property) refer to those elements that a context provider is capable to process but they are not necessarily needed for a successful execution of the context provider's acquisition activities. Those specifications allow for a detailed, element-level orchestration of context providers. A vocabulary specification may contain the namespaces of multiple vocabularies in case the terms of two (or more) vocabularies are being used for the description of the contextual information contained in a context model.

Additionally, a data description specifies the namespaces and terms that the context provider emits as output data (indicated by the `ddesc:output` property). This property is mandatory for all context providers. The output description follows the schema of the input description, consisting of parts for vocabulary, concepts, and properties. In contrast to the input specification, the output specification may consist only of mandatory elements[23].

We use an Extended Backus-Naur Form (EBNF) like notation for representing the main structural aspects of the data description ontology represented by its main classes and properties as listed in Table 4.4 and 4.5. It is to be read as follows: a data description consists of a specification section which itself consists of an optional input section called input specification and exactly one output section called output specification. Both input and output specification consist of a vocabulary specification that by itself consist of a namespace section, a concepts section, and a properties section. Both concepts and properties section consist of a mandatory and an optional section. These sections contain an arbitrary number of terms represented by their URIs. Please note that this notation of the core structural elements defined by the data description ontology is a rather reduced form and disregards the properties as well as relationships and constraints defined between the constituting terms. Instead, this structural (E)BNF-based notation should give the reader an understanding of the basic structure of a context provider's data description represented using the data description ontology.

Figure 4.5 depicts an excerpt of an exemplary data description being defined for a complementary context provider that extracts contact data from acquired calendar data. The complementary

[23] According to the RDF semantics it is possible to specify optional data, although they will not be considered by the orchestration framework in its current version.

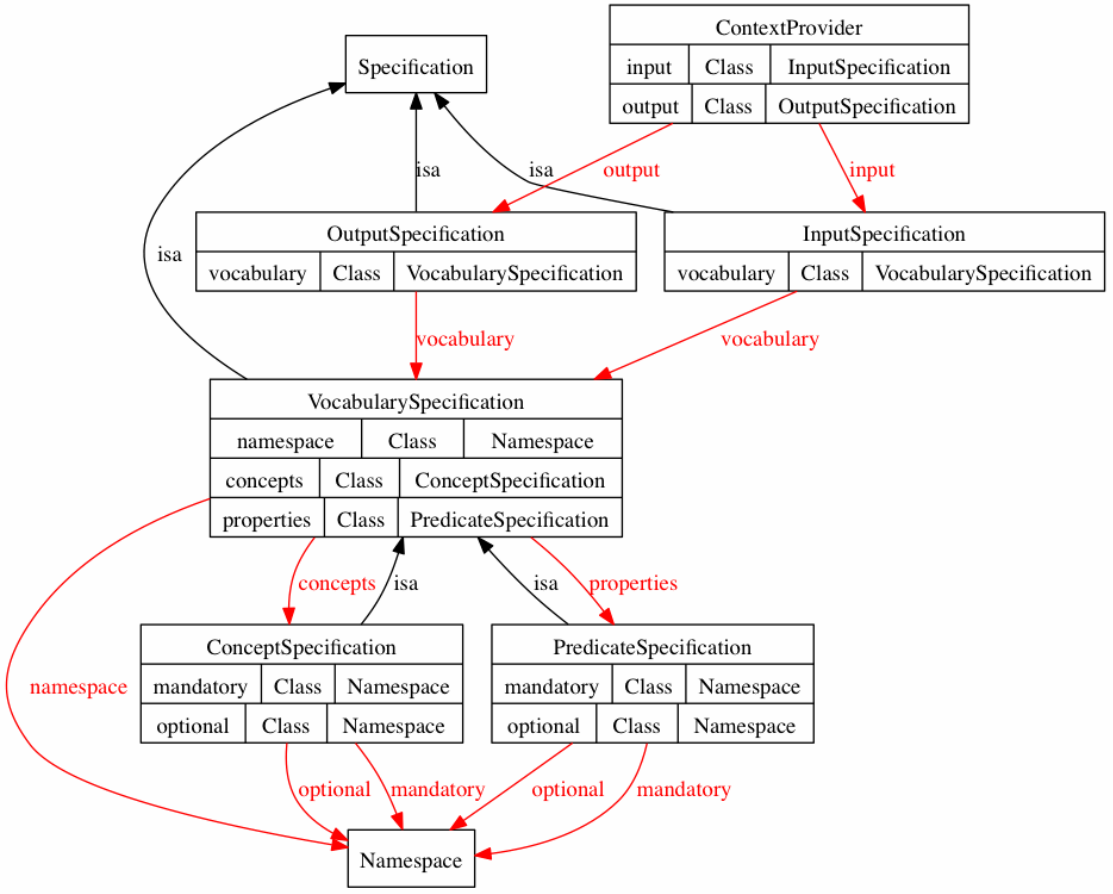


FIGURE 4.4: Structural composition of elements constituting the data description ontology

context provider is capable to process input context models the content of which is described using the NEPOMUK Calendar Ontology[24], indicated by the namespace `http://www.semanticdesktop.org/ontologies/ncal#` in line 5. The terms specified in line 7 are considered mandatory, i.e., the data in the input context model must be described using these terms. In other words, the context provider is only capable to extract contact data from calendar data being described with the specified terms. In addition, the complementary context provider may also process data being described with the terms specified in conjunction with the `ddesc:optional` property in line 8. The lines 10 to 12 describe these facts for the properties being defined in the NEPOMUK Calendar Ontology, which are relevant for the processing of the input model. The elements constituting the context model emitted by the context provider are specified in the lines 16 to 25; the context provider uses elements from the FOAF Ontology[25] to represent the contact data extracted from the calendar items contained in the input context model. As described in the previous paragraph regarding the structure and semantics of the data description elements, the output specification does not specify any optional concepts or properties.

4.4.2 Computing Compatibility Metrics between Context Providers

This section describes the transformation of the context providers' data descriptions into inverted term indices vectors that serve as a basis for computing the overall compatibility scores

[24]NEPOMUK Calendar Ontology: `http://www.semanticdesktop.org/ontologies/ncal/`
[25]FOAF Vocabulary Specification: `http://xmlns.com/foaf/spec/`

```
<urn:uuid:b772a3a2-46d4-4c43-8f71-7080915ddba7>
   a   ddesc:ContextProvider ;
   ddesc:input [
      ddesc:vocabulary [
         ddesc:namespace <http://www.semanticdesktop.org/ontologies/ncal#> ;
         ddesc:concepts [
            ddesc:mandatory ncal:Attendee, ncal:Calendar, ncal:Event ;
            ddesc:optional ncal:Organizer, ncal:EventStatus
         ] ;
         ddesc:properties [
            ddesc:mandatory ncal:member, ncal:method ;
            ddesc:optional ncal:eventStatus
         ]
      ]
   ] ;
   ddesc:output [
      ddesc:vocabulary [
         ddesc:namespace <http://xmlns.com/foaf/0.1/> ;
         ddesc:concepts [
            ddesc:mandatory foaf:Organization, foaf:Person
         ] ;
         ddesc:properties [
            ddesc:mandatory foaf:knows, foaf:status, foaf:name
         ]
      ]
   ] .
```

FIGURE 4.5: Exemplary data description for a complementary context provider for extracting contact data from calendar entries

between the context providers $cp \in CP$. The compatibility scores are recorded in a compatibility matrix $\mathbf{M}^C$ from which the the orchestration matrix $\mathbf{M}^O$ is computed that serves as a basis for deducing the orchestration trees $o_p \in O$. In this section, we therefore describe the basic routines and algorithms underlying this process.

For the following description, we presuppose the existence of the partitions P and C resulting from a classification of the set of context providers CP. Please note that the basic algorithms described in this section do not consider `rdfs:subClassOf` or `rdfs:subPropertyOf` relationships between terms defined in the respective vocabularies. More specifically, further equivalence indicators defined in OWL such as `owl:equivalentClass`, `owl:equivalentProperty`, or `owl:sameAs` are also not considered. Such an advanced form of compatibility analysis taking into account the semantics and logic calculuses from distinct vocabulary elements (used for describing the contextual information represented in a context model) that were defined on the basis of axiomatically defined Semantic Web language elements are out of the scope of our work as this is primarily a branch of ontology alignment (cf. [DMDH02, MBDH02, CSC04]), ontology mapping (cf. [DMD+03, CSH06]), and (semantic) web service alignment respectively.

4.4.2.1 Creation of Inverted Term Indices

For calculating the compatibility score between two context providers $cp_i, cp_j \in CP$, the terms being relevant for the compatibility score calculation defined in the data descriptions are represented in the form of inverted term indices (cf. [MRS08]). In a first step, we therefore create the inverted term indices, denoted as I, for the context providers their compatibility is to be computed by retrieving the terms t specified in the relevant parts of their data descriptions.

Table 4.6: Symbols and descriptions of the inverted term indices

Symbol	Description
$I^{C_{output}}_{mandatory}(cp)$	denotes the inverted term index for the concepts specified in the mandatory part (`ddesc:mandatory`) of the output part (`ddesc:output`) defined in the data description *ddesc* for a context provider $cp \in CP$.
$I^{P_{output}}_{mandatory}(cp)$	denotes the inverted term index for the properties specified in the mandatory part (`ddesc:mandatory`) of the output part (`ddesc:output`) defined in the data description *ddesc* for a context provider $cp \in CP$.
$I^{C_{input}}_{mandatory}(c)$	denotes the inverted term index for the concepts specified in the mandatory part (`ddesc:mandatory`) of the input part (`ddesc:input`) defined in the data description *ddesc* for a complementary context provider $c \in C$.
$I^{P_{input}}_{mandatory}(c)$	denotes the inverted term index for the properties specified in the mandatory part (`ddesc:mandatory`) of the input part (`ddesc:input`) defined in the data description *ddesc* for a complementary context provider $c \in C$.
$I^{C_{input}}_{optional}(c)$	denotes the inverted term index for the concepts specified in the optional part (`ddesc:optional`) of the input part (`ddesc:input`) defined in the data description *ddesc* for a complementary context provider $c \in C$.
$I^{P_{input}}_{optional}(c)$	denotes the inverted term index for the properties specified in the optional part (`ddesc:optional`) of the input part (`ddesc:input`) defined in the data description *ddesc* for a complementary context provider $c \in C$.

Therefore, let I be an ordered list[26] of terms t and defined as an n-tuple $\langle t_1, t_2, \ldots, t_n \rangle$ where n refers to the number of *unique* terms specified in the respective parts of the data description *ddesc* of the corresponding context provider $cp \in CP$:

$$I := \langle t_1, t_2, \ldots, t_n \rangle \tag{4.34}$$

As an inverted term index I is defined on the basis of set theory and as we presume that the vocabularies' terms are defined on the basis of the URI addressing scheme, i.e., each term t is represented by a unique URI, a term t occurs maximally once in I. Therefore, we define the following condition for all terms t represented in an inverted term index I:

$$\forall t \in I \nexists t_i, t_j : t_i = t_j \quad \text{for } i \neq j \tag{4.35}$$

In order to distinguish between the different types of inverted term indices relevant for the compatibility calculation and the terms defined in the respective sections of a data description (cf. Figure 4.3 and Table 4.4), the corresponding part an inverted term index refers to is added as superscript and subscript to the inverted term index symbol I. For instance, an inverted term index of the concepts specified in the output section of a data description is thus denoted as $I^{C_{output}}$ whereas the inverted term index for the properties specified in the input section of a data description is denoted as $I^{P_{input}}$. Moreover, to distinguish whether an inverted term index I refers to a mandatory or optional specification part of a data description, we add the corresponding label as subscript. The mandatory terms specified in the output specification of a data description are thus represented by the inverted term index $I^{C_{output}}_{mandatory}$. Table 4.6 provides an overview of the set of inverted term indices necessary for the compatibility score calculation together with a description of the aspects and data description sections they refer to.

[26] Naturally, the elements of an inverted term index are sorted in alphabetic order.

For example, the inverted term index $I^{P_{output}}_{mandatory}(c)$ represents the mandatory properties specified in the output section of the exemplary data description *ddesc* for a complementary context provider $c \in C$ depicted in Figure 4.5 would be $\langle$'foaf:name','foaf:knows','foaf:status'$\rangle$[27]. This inverted term index shows a real simple case whereas for context providers deployed in productive systems, the corresponding data descriptions are likely to be more complex and comprise an order of magnitude more terms.

In a next step, the inverted term indices I of two context providers $cp_i, cp_j \in CP$ for $i \neq j$ their compatibility score is to be computed are pairwise merged into a compound inverted term index using a two-parametric $f_{merge} : I \times I \rightarrow I$ function; we call this index *merged inverted term index* I^{merged}. The function $f_{merge}\big(I(cp_i), I(cp_j)\big)$ aggregates the terms t of the two inverted term indices $I(cp_i)$ and $I(cp_j)$ into I^{merged} while considering the sequential order of terms t and their identity, i.e., if a term t is an element of both tuples $I(cp_i), I(cp_j)$, t occurs only once in the merged inverted term index $I^{merged}(cp_i, cp_j)$. Therefore, let I^{merged} be an ordered set of terms t and defined as the result of the function $f_{merge}\big(I(cp_i), I(cp_j)\big)$ for the inverted term indices I of two context providers $cp_i, cp_j \in CP$ with compliance of Equation (4.35):

$$I^{merged}(cp_i, cp_j) := f_{merge}\big(I(cp_i), I(cp_j)\big) \tag{4.36}$$

For computing the overall compatibility score between two context providers $cp_i, cp_j \in CP$ for $i \neq j$, the following merged inverted term indices $I^{merged}(cp_i, cp_j)$, represented by Equation (4.37) - (4.40), are created from the inverted term indices $I(cp_i)$ and $I(cp_j)$ of their respective context providers $cp_i, cp_j \in CP$ (cf. Table 4.6):

$$I^{C_{merged}}_{mandatory}(cp_i, cp_j) = f_{merge}\big(I^{C_{output}}_{mandatory}(cp_i), I^{C_{input}}_{mandatory}(cp_j)\big) \tag{4.37}$$

$$I^{P_{merged}}_{mandatory}(cp_i, cp_j) = f_{merge}\big(I^{P_{output}}_{mandatory}(cp_i), I^{P_{input}}_{mandatory}(cp_j)\big) \tag{4.38}$$

$$I^{C_{merged}}_{optional}(cp_i, cp_j) = f_{merge}\big(I^{C_{output}}_{mandatory}(cp_i), I^{C_{input}}_{optional}(cp_j)\big) \tag{4.39}$$

$$I^{P_{merged}}_{optional}(cp_i, cp_j) = f_{merge}\big(I^{P_{output}}_{mandatory}(cp_i), I^{P_{input}}_{optional}(cp_j)\big) \tag{4.40}$$

Depending on whether the merged inverted term index is to be created from the inverted term indices representing the concepts' or properties' section of a data description, these indicators are added as superscripts. Furthermore, the corresponding mandatory or optional sub graphs[28] are added as subscripts.

For example, let T be the set of terms $t \in T$ used in the data descriptions of two context providers $cp_i, cp_j \in CP$ for $i \neq j$ where no distinction is necessary regarding the specific data description sections they refer to. Furthermore, let's assume that $I(cp_i)$ and $I(cp_j)$ contain the following terms $t \in T$ specified in the respective parts of the data descriptions: $I(cp_i) := \langle t_2, t_3, t_6, t_7 \rangle$ and $I(cp_j) := \langle t_1, t_2, t_5, t_7, t_9 \rangle$. The merged inverted term index I^{merged} of the two context providers $cp_i, cp_j \in CP$ is hence the union of the two inverted term indices $I^{merged}(cp_i, cp_j) = \langle t_1, t_2, t_3, t_5, t_6, t_7, t_9 \rangle$.

[27] For reasons of clarity and comprehensibility we use a shorthand notion for the corresponding namespaces.

[28] As data descriptions are represented as RDF graphs, we use the terms 'section', 'sub section', and 'sub graph' synonymously.

4.4.2.2 Transformation into Inverted Term Index Vectors

For the compatibility scores calculations, we use a *multi-dimensional vector space model* in which an inverted term index $I(cp_i)$ of a context provider $cp_i \in CP$ and thus the terms t specified in the relevant part of its data description (see Table 4.6) are transformed into a vector representation $\langle v_1, v_2, \ldots, v_n \rangle \in \{0,1\}^n$ that serves as a basis for deriving the *inverted term index vectors* of both context providers $cp_i, cp_j \in CP$ their compatibility score is to be computed. As an inverted term index I[29] represents an ordered sequence of terms t, it can be represented by the binary inverted term indices vector $\vec{V} = \langle v_{t_1}, v_{t_2}, \ldots, v_{t_{|I^{merged}|}} \rangle \in \{0,1\}^{|I^{merged}|}$ where each vector component v_t refers to a term t being a member of the merged inverted term index $I^{merged}(cp_i, cp_j)$ representing the union of the inverted term indices $I(cp_i)$ and $I(cp_j)$. Therefore, let $\vec{V}^{index}(cp_i)$ denote the inverted term index vector calculated from an inverted term index $I^{index}(cp_i)$ of the corresponding context provider $cp_i \in CP$ and the merged inverted term index $I^{merged}(cp_i, cp_j)$ for $cp_i, cp_j \in CP$ where $i \neq j$[30]:

$$\vec{V}^{index}(cp_i) := \left\langle v_{t_1}, v_{t_2}, \ldots, v_{t_{|I^{merged}(cp_i,cp_j)|}} \right\rangle \in \{0,1\}^{|I^{merged}(cp_i,cp_j)|} \tag{4.41}$$

For determining the value of a vector component v_t, we transform inverted term indices I^{index} to a set notation that allows for computing the intersection between the terms t contained in both the inverted term index $I(cp_i)$ and the merged inverted term index $I^{merged}(cp_i, cp_j)$. Therefore, let $T_{I^{index}}(cp_i)$ and $T_{I^{merged}}(cp_i, cp_j)$ include those terms t, that are contained in the respective inverted term indices. A term t is a member of T_{index}, iff the predicate $containedIn(term, index)$ evaluates to true:

$$T_{I^{index}}(cp_i) := \left\{ t : containedIn\big(t, I^{index}(cp_i)\big) \right\} \tag{4.42}$$

$$T_{I^{merged}}(cp_i, cp_j) := \left\{ t : containedIn\big(t, I^{merged}(cp_i, cp_j)\big) \right\} \tag{4.43}$$

On the basis of Equation (4.42) and (4.43), the value of a vector component v_t of an inverted term index vector $\vec{V}^{index}(cp)$ is defined by the following equation:

$$v_t := \begin{cases} 1 & : \quad t \in \big(T_{I^{index}}(cp_i) \cap T_{I^{merged}}(cp_i, cp_j)\big) \\ 0 & : \quad \text{otherwise} \end{cases} \tag{4.44}$$

The value of each vector component v_t indicates the existence of a term t specified in the inverted term index $I(cp)$ a vector $\vec{V}(cp)$ refers to. A vector component $v_t = 1$ in case the term t is contained in the inverted term index $I(cp)$ of the corresponding context provider $cp \in CP$ and $v_t = 0$ in case the term t is not present in $I(cp)$. Furthermore, the sequence of the vector components v_t corresponds to the sequence of terms t specified in the respective merged inverted term index $I^{merged}(cp_i, cp_j)$ and the respective inverted term index $I(cp_i)$ for cp_i. A term t thus serves as index for a vector component and refers to the component v_t corresponding to t. For instance, the inverted term index vector of an inverted term index $I(cp_i)$ for a context provider cp_i from the previous example is thus $\vec{V}(cp_i) = \langle t_1 : 0,\ t_2 : 1,\ t_3 : 1,\ t_5 : 0,\ t_6 : 1,\ t_7 : 1,\ t_9 : 0 \rangle$ as only those terms t, where the value of the corresponding vector components v_t is 1, are contained in the inverted term index $I(cp_i)$.

[29] For reasons of clarity and comprehensibility we omit the use superscripts and subscripts for the explanations given in this paragraph as these deliberations apply to all types of inverted term indices I likewise.

[30] For the general definition of the vector representation $\vec{V}$ of an inverted term index I, the use of superscripts and subscripts usually applied to the symbols of inverted term indices and their respective vector representations has been omitted; instead the superscript ^{index} is used for both symbol I and $\vec{V}$ to indicate the supplementation with the respective indices.

Referring to Equation (4.37) - (4.40), an inverted term index vector $\vec{V}^{index}$ is created for every inverted term index I outlined in Table 4.6. The rules for the use of superscripts and subscripts expounded in Section 4.4.2.1 also apply for the symbols of the inverted term index vectors. In consequence, let $\vec{V}^{C_{output}}_{mandatory}(cp)$ be the vector created from the inverted term index $I^{C_{output}}_{mandatory}$ of the concepts specified in the mandatory section of the data description *ddesc* of the context provider $cp \in CP$ and let $\vec{V}^{C_{input}}_{mandatory}(cp)$, $\vec{V}^{P_{output}}_{mandatory}(cp)$, $\vec{V}^{P_{input}}_{mandatory}(c)$, $\vec{V}^{C_{input}}_{optional}(cp)$, and $\vec{V}^{P_{input}}_{optional}(cp)$ be the inverted term index vectors for the other types of inverted term indices. Hence, each vector symbol corresponds to exactly one inverted term index symbol defined in Table 4.6.

4.4.2.3 Calculation of Compatibility Scores

A well-known and standard way in information retrieval for quantifying the similarity between different documents is to compute the cosine similarity of their vector representations (cf. [MRS08]). However, as relevant sections of a context provider's data description are represented as inverted term indices, we can adopt this methodology for the calculation of a similarity score between two context providers $cp_i, cp_j \in CP$ for $i \neq j$ on the basis of the vector representations $\vec{V}^{index}$ of their inverted term indices I^{index}. The cosine similarity calculation formula is expressed in Equation (4.45)[31] where n refers to the number of vector components v_t constituting an inverted term index vector $\vec{V}^{index}(cp)$ for a context provider $cp \in CP$:

$$\mathrm{sim}\big(\vec{V}(cp_i), \vec{V}(cp_j)\big) = \frac{\vec{V}(cp_i) \cdot \vec{V}(cp_j)}{\big|\vec{V}(cp_i)\big|\big|\vec{V}(cp_j)\big|} = \frac{\sum_{k=1}^{n} \vec{V}(cp_i)_k \times \sum_{k=1}^{n} \vec{V}(cp_j)_k}{\sqrt{\sum_{k=1}^{n} \Big(\vec{V}(cp_i)_k\Big)^2} \times \sqrt{\sum_{k=1}^{n} \Big(\vec{V}(cp_j)_k\Big)^2}} \quad (4.45)$$

For each combination of context providers $cp_i, cp_j \in CP$ where $i \neq j$ their overall compatibility is to be determined, four different compatibility scores are computed using Equation (4.45), as defined by the Equation (4.46) - (4.49):

$$\mathrm{sim}\big(\vec{V}^{C_{output}}_{mandatory}(cp_i), \vec{V}^{C_{input}}_{mandatory}(cp_j)\big) \quad (4.46)$$

$$\mathrm{sim}\big(\vec{V}^{P_{output}}_{mandatory}(cp_i), \vec{V}^{P_{input}}_{mandatory}(cp_j)\big) \quad (4.47)$$

$$\mathrm{sim}\big(\vec{V}^{C_{output}}_{mandatory}(cp_i), \vec{V}^{C_{input}}_{optional}(cp_j)\big) \quad (4.48)$$

$$\mathrm{sim}\big(\vec{V}^{P_{output}}_{mandatory}(cp_i), \vec{V}^{P_{input}}_{optional}(cp_j)\big) \quad (4.49)$$

These compatibility scores are recorded in a compatibility vector $\vec{V}_{Compatibility}(cp_i, cp_j)$ that contains the compatibility scores (cosine similarities) calculated between the vector representations of the inverted term indices of mandatory concepts and properties specified in the output and input sections as well as for the terms specified in the optional sections of the input specification (cf. Section 4.4.1). The vector component v_1 hence represents the cosine similarity score computed by Equation (4.46), v_2 the score computed by Equation (4.47) and so forth.

Furthermore, a compatibility vector $\vec{V}_{Compatibility}(cp_i, cp_j)$ contains two additional components $v_5 = \theta^C \in \{0, 1\}$ and $v_6 = \theta^P \in \{0, 1\}$ indicating the degree of overlap between the inverted term indexes of cp_i and cp_j regarding their mandatory concepts and properties (cf. Equation (4.37)

[31] For readability reasons, we omitted the use of indices for the vector representations of the inverted term indices in Equation (4.45).

and (4.38)). For the calculation of θ^C and θ^P, we use an adapted notation as defined in Equation (4.42) and (4.43) as these allow us to compute the degree of term intersection between two inverted term indices. Therefore, let the predicate $containedIn(term, index)$ qualify those terms t that are contained in an inverted term index:

$$\theta^C = \begin{cases} 1 & \iff \forall t\ containedIn\big(t, I^{C_{input}}_{mandatory}(cp_j)\big) : containedIn\big(t, I^{C_{output}}_{mandatory}(cp_i)\big) \\ 0 & \text{otherwise} \end{cases} \tag{4.50}$$

$$\theta^P = \begin{cases} 1 & \iff \forall t\ containedIn\big(t, I^{P_{input}}_{mandatory}(cp_j)\big) : containedIn\big(t, I^{P_{output}}_{mandatory}(cp_i)\big) \\ 0 & \text{otherwise} \end{cases} \tag{4.51}$$

A value of 1 indicates that all terms t defined in the input part of the data description of cp_j are contained in the output part of the data description of cp_i. More specifically, $\theta^C = 1$ signals that all the concepts specified in $I^{C_{input}}_{mandatory}(cp_j)$ are also specified in $I^{C_{output}}_{mandatory}(cp_i)$ and therefore the result of the cosine similarity calculation $\text{sim}\big(\vec{V}^{C_{output}}_{mandatory}(cp_i), \vec{V}^{C_{input}}_{mandatory}(cp_j)\big)$ is to be interpreted as a complete conformity and full compatibility respectively. This indicator is necessary as the concepts' or properties' specifications of two context providers $cp_i, cp_j \in CP$ might be completely compatible although their compatibility score is < 1, which is the case when, for instance, the output specification of either concepts or properties of cp_i is more extensive than the input specification of cp_j, i.e., the set of terms required by a context provider $cp_j \in CP$ to be used in the context model $m \in M_{cp_i}$ it consumes is a real subset of the set of terms a context provider $cp_i \in CP$ uses to describe the contextual information contained in the context model $m \in M_{cp_i}$ it emits.

In consequence, a compatibility vector $\vec{V}_{Compatibility}(cp_i, cp_j)$ containing the compatibility scores computed from the vector representations $\vec{V}^{index}(cp_i)$ and $\vec{V}^{index}(cp_i)$ of the inverted term indices of two context providers $cp_i, cp_j \in CP$ consists of six vector components, as defined in Equation (4.46) - (4.51), and is defined as follows[32]:

$$\vec{V}_{Compatibility}(cp_i, cp_j) = \begin{bmatrix} \text{sim}\big(\vec{V}^{C_{output}}_{mandatory}(cp_i), \vec{V}^{C_{input}}_{mandatory}(cp_j)\big) \\ \text{sim}\big(\vec{V}^{P_{output}}_{mandatory}(cp_i), \vec{V}^{P_{input}}_{mandatory}(cp_j)\big) \\ \text{sim}\big(\vec{V}^{C_{output}}_{mandatory}(cp_i), \vec{V}^{C_{input}}_{optional}(cp_j)\big) \\ \text{sim}\big(\vec{V}^{P_{output}}_{mandatory}(cp_i), \vec{V}^{P_{input}}_{optional}(cp_j)\big) \\ \theta^C \\ \theta^P \end{bmatrix} \tag{4.52}$$

For instance, a compatibility vector $\vec{V}_{Compatibility}(cp_i, cp_j)$ containing the similarity scores for two context providers $cp_i, cp_j \in CP$ for $i \neq j$ could consist of the following compatibility scores:

$$\vec{V}_{Compatibility}(cp_i, cp_j) = [0.707 \quad 0.982 \quad 0.128 \quad 0.000 \quad 1 \quad 0]$$

With reference to Equation (4.46), the first value represents the compatibility score computed for the mandatory concepts specified in the input and output sections of the data descriptions of $cp_i, cp_j \in CP$. A value of 0.707 indicates that both context providers specify a different set of terms t in the corresponding sections of their data description, i.e., there exists no full correspondence between the terms t contained in $I^{C_{input}}_{mandatory}(cp_j)$ and $I^{C_{output}}_{mandatory}(cp_i)$. However as $\theta^C = 1$, the compatibility score represented by v_1 is to be interpreted as a full compatibility

[32]For reasons of clarity and comprehensibility, the vector components of the compatibility vector $\vec{V}_{Compatibility}(cp_i, cp_j)$ are specified in a column vector representation in Equation (4.52).

regarding the corresponding inverted term indices although the calculated similarity score of their vector representations is < 1 . The value of the second vector component $v_2 = 0.982$ indicates that both context providers specify different sets of properties and as $\theta^P = 0$, $I^{P_{input}}_{mandatory}(cp_j) \nsubseteq I^{P_{output}}_{mandatory}(cp_i)$ wherefore this compatibility score is *not* to be interpreted as full compatibility regarding the specification of mandatory properties. The vector components $v_3 = 0.128$ and $v_4 = 0.000$ represent the compatibility scores calculated for the optional concepts and properties cp_j is able to process.

The individual values of the compatibility vector components are then aggregated into a final compound compatibility score that quantifies the overall degree of compatibility of two context providers $cp_i, cp_j \in CP$ and is calculated on the basis of a linear combination of the single compatibility scores recorded in the corresponding compatibility vector $\vec{V}_{Compatibility}(cp_i, cp_j)$. The final compatibility scores are computed for all possible combinations, i.e., pairs of context providers $cp_i, cp_j \in CP$ for $i \neq j$ and are recorded in the compatibility matrix $\mathbf{M}^C$. However, there exist plenty of possible calculation heuristics that might be adapted to meet the requirements of domain or application-specific settings. In the following, we outline a simple method for calculating the overall compatibility score between two context providers $cp_i, cp_j \in CP$ for $i \neq j$ on the basis of a linear combination of the arithmetic means of the corresponding compatibility vector's components (cf. Equation (4.52)) for the case that both θ^C and θ^P are 0:

$$\frac{\text{sim}\big(\vec{V}^{C_{output}}_{mandatory}(cp_i), \vec{V}^{C_{input}}_{mandatory}(cp_j)\big) + \text{sim}\big(\vec{V}^{P_{output}}_{mandatory}(cp_i), \vec{V}^{P_{input}}_{mandatory}(cp_j)\big)}{2} \quad \text{iff } \theta^C, \theta^P = 0$$

In case $\theta^C(cp_i, cp_j) = 1$, the compatibility score $\text{sim}\big(\vec{V}^{C_{output}}_{mandatory}(cp_i), \vec{V}^{C_{input}}_{mandatory}(cp_j)\big)$ is replaced by the value 1; for $\theta^P(cp_i, cp_j) = 1$, $\text{sim}\big(\vec{V}^{P_{output}}_{mandatory}(cp_i), \vec{V}^{P_{input}}_{mandatory}(cp_j)\big) = 1$ in the equation above. Iff there is a full compatibility w.r.t. the mandatory scores (cf. Equation (4.46) and (4.47)) a possible solution is to further consider the scores computed for the optional terms (cf. Equation (4.48) and (4.49)) to further differentiate between identical compatibility scores computed on the basis of the mandatory terms' specification. This allows for selecting and orchestrating the most appropriate context provider, i.e., the context provider with the highest overall score from a set of context providers that have identical compatibility scores computed on the basis of their mandatory terms for a given context provider. The following equation depicts a simple calculation heuristic that integrates the optional compatibility scores into the calculation of the overall compatibility score between two context providers $cp_i, cp_j \in CP$ for $i \neq j$ iff $\theta^C = \theta^P = 1$:

$$1 + \frac{\text{sim}\big(\vec{V}^{C_{output}}_{mandatory}(cp_i), \vec{V}^{C_{input}}_{optional}(cp_j)\big) + \text{sim}\big(\vec{V}^{P_{output}}_{mandatory}(cp_i), \vec{V}^{P_{input}}_{optional}(cp_j)\big)}{2} \quad \text{iff } \theta^C, \theta^P = 1$$

The above equation represents the calculation for the inverse case where $\theta^C = \theta^P = 1$ and has to be adapted accordingly in case only one of the two subset indicators θ^C or θ^P is 1 or in case the value of both indicators is 0. The final result of the compatibility score calculation is then integrated in the compatibility matrix $\mathbf{M}^C$ as a matrix coefficient of the row and column indexes representing the two context providers cp_i and cp_j.

4.4.2.4 Creation of Compatibility Matrix $\mathbf{M}^C$ and Orchestration Matrix $\mathbf{M}^O$

The compatibility matrix $\mathbf{M}^C$ contains the final compatibility scores of all possible pairs of context providers $cp_i, cp_j \in CP$. It is defined as a $|CP| \times |CP|$-matrix composed from the set of all context providers $cp \in CP$ deployed in a running instance of the context framework. The

matrix coefficient of a row i and a column j indicates the degree of compatibility computed for the context provider $cp_i \in CP$ represented by the i-th row and the context provider $cp_j \in CP$ represented by the j-th column. A matrix coefficient $\mathbf{M}^C[cp_i, cp_j]$ thus represents the overall compatibility score computed for the two context providers cp_i and cp_j where cp_i is the emitting and cp_j the receiving context provider of a context model $m \in M_{cp_i}$. According to the compatibility score calculation method introduced in Section 4.4.2.3, a matrix coefficient $\mathbf{M}^C[i,j] >= 1$ indicates a full compatibility between the context provider $cp_i \in CP$ represented in the i-th row and the context provider $cp_j \in CP$ represented by the j-th column of the compatibility matrix $\mathbf{M}^C$. This means that the context model $m \in M_{cp_i}$ emitted by cp_i contains at minimum all mandatory concepts and properties cp_j specified in the input section of its data description and requires the contextual information contained in the context model $m \in M_{cp_i}$ to be described with for its further refinement and complementation. In such a case, the corresponding matrix coefficient in the orchestration matrix $\mathbf{M}^O$ is set to 1 as a relation r can be established between them (cf. Section 4.3.4). A matrix coefficient $\mathbf{M}^C[i,j] < 1$ signals that two context providers $cp_i, cp_j \in CP$ are not entirely compatible as there is a mismatch or misalignment between the set of terms they regard mandatory in the output and input sections of their data descriptions.

The orchestration matrix $\mathbf{M}^O$ is defined as a $|CP| \times |CP|$ adjacency matrix which specifies the relations $r \in R$ existing between the context providers $cp \in CP$ deduced from the compatibility matrix $\mathbf{M}^C$ where R is the superset of all relations that can be deduced from $\mathbf{M}^O$ (see Equation (4.54)). A matrix coefficient $\mathbf{M}^O\big[\alpha(r_i), \omega(r_i)\big]$ between two context providers $\alpha(r_i)$, $\omega(r_i)$ being incident to a relation $r_i \in R$ is 1 in case a relation can be established between them as a result of an analysis of their compatibility score. The coefficient value 0 indicates that no relation $r \in R$ exists between two context providers $\alpha(r), \omega(r) \in CP$ which is to be interpreted as an incompatibility.

After the compatibility matrix $\mathbf{M}^C$ has been created, it is analyzed and transformed into the orchestration matrix $\mathbf{M}^O$; Algorithm 1 describes this process. The algorithm requires a compatibility matrix $\mathbf{M}^C$ and a threshold value that determines the value at which two context providers $cp_i, cp_j \in CP$ are considered compatible as input parameters. A common threshold value is 1.

Algorithm 1: Transforming the compatibility matrix $\mathbf{M}^C$ into the orchestration matrix $\mathbf{M}^O$

Data: Compatibility Matrix $\mathbf{M}^C$, Threshold *threshold*
Result: Orchestration Matrix $\mathbf{M}^O$

```
1  Initialize(M^O)                                   /* set all coefficients to 0 */
2  for i = 1 to |CP| do
3  |  for j = 1 to |CP| do
4  |  |  if M^C[i,j] >= threshold then
5  |  |  |  if (M^C[1...|CP|,j]_max < M^C[i,j]) AND (sum_{k=1}^{i} M^O[k,j] < 1) then
6  |  |  |  |  M^O[i,j] ← 1
7  |  |  |  end
8  |  |  end
9  |  end
10 end
```

In a first step, the orchestration matrix $\mathbf{M}^O$ is initialized by setting the values of all its matrix coefficients to 0. For each possible combination of context providers $cp_i, cp_j \in CP$, we check whether the value of the corresponding matrix coefficient indicated by the i-th row and j-th column is equal to or above the predefined threshold. If this is the case, the condition in line 5 assesses whether the current coefficient $\mathbf{M}^C[i,j]$ exhibits the greatest value in the current column j of the compatibility matrix $\mathbf{M}^C[1 \ldots |CP|, j]$ and whether there does not exist any

$$\begin{bmatrix} 0 & 0 & 0 & 1.02 & 1.73 & 0.24 & 0.00 & 0.08 \\ 0 & 0 & 0 & 0.45 & 0.03 & 0.56 & 0.88 & 1.04 \\ 0 & 0 & 0 & 0.02 & 0.01 & 0.01 & 0.00 & 0.01 \\ 0 & 0 & 0 & 0 & 0.05 & 0.12 & 0.01 & 0.13 \\ 0 & 0 & 0 & 0.02 & 0 & 1.44 & 1.00 & 0.77 \\ 0 & 0 & 0 & 0.45 & 0.09 & 0 & 0.01 & 0.01 \\ 0 & 0 & 0 & 0.10 & 0.01 & 0.03 & 0 & 0.68 \\ 0 & 0 & 0 & 0.02 & 0.17 & 0.06 & 0.01 & 0 \end{bmatrix} \Longrightarrow \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

FIGURE 4.6: Example of a compatibility matrix $\mathbf{M}^C$ and the corresponding orchestration matrix $\mathbf{M}^O$

coefficient value > 0 in the corresponding column of the orchestration matrix $\mathbf{M}^O[1 \dots |CP|, j]$. If both conditions evaluate to true, the corresponding coefficient in the orchestration matrix $\mathbf{M}^O[i, j]$ is set to 1, otherwise the matrix coefficient remains in its default value 0. The value 0 indicates no compatibility at all and is per definition set for identical context providers or between primary context providers $p \in P$ as those per definition (cf. Definition 4.3 and 4.4) can not mutually complement or refine their context models $m \in M$. However, in case no overall compatibility score between two context providers $cp_i, cp_j \in CP$ was computed, i.e., no compatibility vector $\vec{V}_{Compatibility}(cp_i, cp_j)$ exists, the corresponding matrix coefficient $\mathbf{M}^O[cp_i, cp_j]$ is set to 0 per default. Figure 4.6[33] depicts an exemplary compatibility matrix $\mathbf{M}^C$ and the corresponding orchestration matrix $\mathbf{M}^O$ generated using the steps defined in Algorithm 1 for three primary context providers $p_1, p_2, p_3 \in P$ and five complementary context providers $c_1, c_2, c_3, c_4, c_5 \in C$, from which the three orchestration trees $o_{p_{1,2,3}} \in O$ depicted in Figure 4.8 can be deduced.

4.4.3 Building Orchestration Trees

As formally defined in the previous section for building the orchestration trees, the context framework analyzes the data description of each context provider $cp \in CP$ and computes a compatibility matrix $\mathbf{M}^C$ that serves as a basis for deducing an orchestration matrix $\mathbf{M}^O$ that defines the adjacency relationships between compatible context providers $p, c \in V_p$ on the basis of the compatibility scores. In this section, we formally define the graph G represented by the orchestration matrix $\mathbf{M}^O$. Therefore, let $G(V, R)$ denote the graph represented by the orchestration matrix $\mathbf{M}^O$ where V represents the set of context providers $cp \in CP$ that exhibit relations to other context providers being members of the set CP and thus can be orchestrated in an orchestration tree $o_p \in O$. Therefore, $V(G)$ is defined as the union of all sets V_p for all $o_p \in O$ and $p \in P$:

$$V(G) := \bigcup_{p \in P} V_p \tag{4.53}$$

$R(G)$ denotes the set of relations $r \in R$ existing for the graph $G(V, R)$ that can be derived from the compatibility matrix $\mathbf{M}^C$ and that exist between compatible context providers $cp \in V, CP$. Similar to Equation (4.53), $R(G)$ is defined as the union of all sets R_p for all $o_p \in O$ and $p \in P$:

$$R(G) := \bigcup_{p \in P} R_p \tag{4.54}$$

[33] For reasons of clarity and comprehensibility we have rounded the compatibility scores to two digits after the period.

As a consequence, every orchestration tree $o_p \in O$ is a partition of the full orchestration graph $G(V, R)$ of all orchestration trees $o_p \in O$ and all context providers $p, c \in V_p$ orchestrated within them. Therefore, all orchestration trees $o_p \in O$ can be regarded as partitions of the set of nodes $V(G)$ of the full orchestration graph $G(V, R)$ with $V_{p_1}, \ldots, V_{p_{|O|}} \subseteq V(G)$ and $V_{p_i} \cap V_{p_j} = \emptyset$ for $i \neq j$. Consequently, an orchestration tree $o_p \in O$ is a subgraph $G[V_p]$ of the graph $G(V, R)$ being induced through the set V_p where $o_p \in O \sqsubseteq G$.

Under this consideration, we can conceive an instance of a context configuration $cc \in CC$ as the result of an aggregation or consolidation function $f_{aggregate} : G(V, R) \longrightarrow OM$ over the entire graph $G(V, R)$ with $V(G) := \bigcup_{i=1}^{|P|} V_{p_i}$ and $R(G) := \bigcup_{i=1}^{|P|} R_{p_i}$ that merges and consolidates the context models emitted by the context providers $c, p \in V$ being orchestrated in the orchestration trees $o_p \in O$, which together constitute the full orchestration graph $G(V, R)$.

Algorithm 2: Deducing the orchestration trees $o_p \in O$ from the orchestration matrix $\mathbf{M}^O$

Data: Orchestration Matrix $\mathbf{M}^O$
Result: Deduction and initialization of orchestration trees $o_p \in O$ for all $p \in P$

```
1  foreach p ∈ P do
2  |  V_p ← { } ; R_p ← { } ;
3  |  V_p ← V_p ∪ {p} ;
4  |  Instantiate(o_p ∈ O) ;
5  end
6  for j = 1 to |CP| do                          // Iterate over all columns in M^O
7  |  for i = 1 to |CP| do                       // Iterate over all rows in M^O
8  |  |  if M^O[i, j] == 1 then
9  |  |  |  R_p ← R_p ∪ {(cp[i], cp[j])} ;      // Create and add tuple r = (cp[i], cp[j])
10 |  |  |  Traverse (i, j) ;
11 |  |  end
12 |  end
13 end
```

Function Traverse(row i, initial index z)

Data: row index i, index of context provider to add z
Result: Add $cp[z]$ to the set $V_{cp[i]}$ of its corresponding primary context provider $cp[i]$

```
1  if cp[i] ∈ P then
2  |  V_cp[i] ← V_cp[i] ∪ {cp[z]} ;      // Add cp[z] to V_cp[i] of its primary context provider
3  else
4  |  for k = 1 to |CP| do
5  |  |  if M^O[k, i] == 1 then
6  |  |  |  Traverse (k, z) ;                    // Traverse M^O until cp[k] ∈ P
7  |  |  end
8  |  end
9  end
```

The adjacency relationships represented by the orchestration matrix $\mathbf{M}^O$ serves as a basis for deducing the orchestration trees $o_p \in O$ of primary context providers $p \in P$ and compatible complementary context providers $c \in C$ where one $o_p \in O$ is deduced for each $p \in P$. The orchestration matrix $\mathbf{M}^O$ contains a separate row and column for every element $p \in P$ and $c \in C$. Algorithm 2 describes the necessary steps for deducing the orchestration trees $o_p \in O$ from the orchestration matrix $\mathbf{M}^O$. Since every row respectively column pertains to one distinct context provider $cp \in CP$, we make use of a *lookup function* $cp[i]$ that refers to the context provider $cp \in CP$ represented by the index i.

	p_1	p_2	p_3	c_1	c_2	c_3	c_4	c_5
p_1	0	0	0	1	1	0	0	0
p_2	0	0	0	0	0	0	0	1
p_3	0	0	0	0	0	0	0	0
c_1	0	0	0	0	0	0	0	0
c_2	0	0	0	0	0	1	1	0
c_3	0	0	0	0	0	0	0	0
c_4	0	0	0	0	0	0	0	0
c_5	0	0	0	0	0	0	0	0

$$\equiv \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

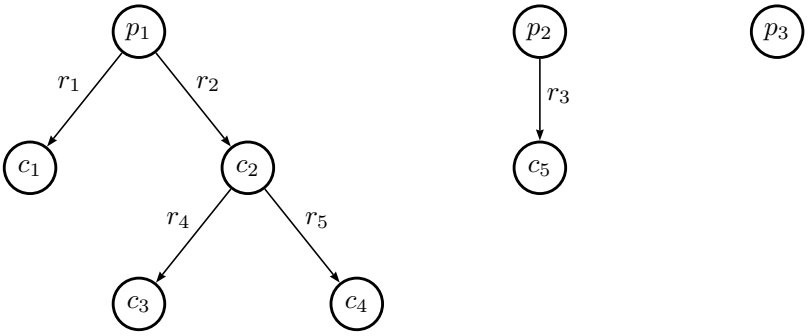
FIGURE 4.7: Example of an adjacency tableau and corresponding orchestration matrix $\mathbf{M}^O$ for eight context providers $p_{1,\ldots,3} \in P$ and $c_{1,\ldots,5} \in C$

In a first step, we instantiate a new instance of an orchestration tree $o_p \in O$ for every primary context provider $p \in P$ deployed in a running instance of the context framework and initialize the sets V_p and R_p; moreover, p is added to the set of constituting context providers V_p (line 1 to 5). On each position where the matrix coefficient $\mathbf{M}^O[i,j] == 1$, a new tuple r consisting of the context provider $cp[i]$, represented by row index i, and context provider $cp[j]$, represented by the column index j, is created and added to R_p (line 10). The algorithm furthermore evaluates whether the context provider $cp[i]$ represented in the current row i is a member of the set of primary context providers P or a member of the set of complementary context providers C (line 1-2 in Function `Traverse`[34]). In case $cp[i]$ is a member of the set of primary context providers P, the context provider $cp[z] \in CP$ represented in the z-th row becomes a member of the set $V_{cp[i]}$ of the corresponding primary context provider $cp[i]$. If $cp[i]$ is not a member of P, the algorithm memorizes the column index of the initial context provider in a variable z, sets the new column index to the value of the corresponding row index i, and recursively reinitiates the traversing process until a new matrix coefficient with value 1 is discovered (line 4 to 8 in Function `Traverse`). Basically, the row index i of the current matrix coefficient $\mathbf{M}^O[i,j]$ becomes the new column index in case the context provider represented in the i-th row is not a member of P. This allows us to retrieve all the context providers $c \in C$ that constitute the set $C_p \subset V_p$ of a primary context provider $p \in P$. However, the algorithm for deducing the orchestration trees $o_p \in O$ from the orchestration matrix $\mathbf{M}^O$ requires the existence of a fixed assignment of context providers $cp \in CP$ and indices.

Figure 4.7 depicts an example of an orchestration matrix containing three primary context providers $p_{1,\ldots,3} \in P$ and five complementary context provider $c_{1,\ldots,5} \in C$. Each row represents a particular context provider $cp \in CP$ where the matrix coefficients in each row indicate a relationship to the context provider given in each column in case its value is set to 1. Likewise, 0 indicates no relationship $r \in R(G)$ between the context provider of the current row and the context provider of the current column. As expounded in Definition 4.3 and 4.4, primary context providers $p \in P$ can not be connected among each other, i.e., primary context providers $p \in P$ always have complementary context providers $c \in C$ as successors. Complementary context providers $c \in C$ in contrast can be connected among each other in a non-cyclic way.

As an example, let there be three primary context providers $p_{1,\ldots,3}$ and five complementary context providers $c_{1,\ldots,5}$ deployed within a running instance of the framework. By analyzing the data descriptions exposed by each context provider (cf. Section 4.4.1), the adjacency tableau and corresponding orchestration matrix $\mathbf{M}^O$ depicted in Figure 4.7 can be created based on the compatibility indicators computed using the methodology expounded in Section 4.4.2. The first

[34]The function `Traverse` is part of Algorithm 2 but specified separately for reasons of comprehensibility and recursive requests.



(a) Orchestration Tree $o_{p_1} \in O$ (b) Orchestration Tree $o_{p_2} \in O$ (c) Orchestration Tree $o_{p_3} \in O$

FIGURE 4.8: Orchestration trees o_{p_1}, o_{p_2}, and o_{p_3} derived from the orchestration matrix $\mathbf{M}^O$ depicted in Figure 4.7

three rows represent those complementary context providers $c \in C$ that are adjacent to the three primary context providers p_1, p_2, and p_3 and take their context models as input. For instance, p_1 has two complementary context providers c_1 and c_2 as direct successors that refine and augment the context model $m \in M_{p_1}$ delivered by $p_1 \in P$.

The values in row 4 show the direct successors of the first complementary context provider c_1. Since the matrix coefficients in the entire row are set to 0, there exist no relationships to other complementary context providers. c_2 in line 5 for instance has two adjacent context providers c_3 and c_4, which both do not have any further successors that take their context models as input for their acquisition tasks. By analyzing the given orchestration matrix $\mathbf{M}^O$, the orchestration trees $o_p \in O$ depicted in Figure 4.8 can be derived for each primary context provider $p_{1,\ldots,3} \in P$[35] using the steps outlined in Algorithm 2. Orchestration trees in general represent atomic units of context acquisition workflow schemes $wf \in WF$. In Section 4.5 we propose a working model for processing the context providers orchestrated in an orchestration tree $o_p \in O$ respectively the data they create and emit.

4.5 A Transaction-based Processing Model for Context Acquisition Workflows

After having introduced the formal model of the context-dependent RDF data replication framework together with a formal description of the orchestration logic and an algorithmic description of the orchestration process, we now present a transaction-based processing model for the acquisition of contextual information, the aggregation of such information into compound context acquisition models, and their transformation into a context configuration that represents the user's current context. The transaction-based model builds on the previous elaborations undertaken in the formal model and guarantees context consistency, accurateness, and completeness while taking into account mobile information system's peculiarities (see [FZ94, KLO+04]). We define the transaction-based processing model on the basis of a formalism developed for the formal description of extended transaction model properties[36] discussed in [CR91a, CR91c] and

[35]The orchestration tree $o_3 \in O$ for context provider $p_3 \in P$ can be considered a special case of an orchestration tree that only contains a root element and no further adjacent elements. This case is discussed in Section 4.3.5

[36]Extended refers to the various models proposed as extensions to the traditional transaction model.

denoted as *ACTA*[37]. One significant problem of the basic transaction model adopted by many traditional databases is that it does not scale in terms of both functionality and performance when an application consists of multiple distributedly operating activities that collaborate in an autonomous and self-contained manner [CR94]. ACTA, among other frameworks proposed for the axiomatic description of extended transaction model properties, is a first-order logic-based formalism that takes into account the nature of interactions existing between extended transactions and allows for characterizing their semantics, the types of dependencies existing between them, as well as their effects on involved transactional objects [CR90, CR94]. For the formal definition of the transactional processing model, however, we use it's core elements as those provide the necessary expressiveness to define the constituting elements with the compellable precision rather than making use of ACTA's full set of description primitives.

In the remainder of this section, we first outline some preliminaries which we consider fundamental for the transaction-based processing model presented in this section. We further define the main elements of the model, provide definitions of the dependencies that exists between the transactions constituting the model, and outline conditions being relevant for the transaction management primitives proposed by our model in an axiomatic way. We complete this section with a definition of the algorithms that are responsible for the acquisition and aggregation of the contextual information represented in the context models emitted by context providers for building a new instance of the context configuration which we conceive as a transit RDF-based manifestation of the user's current context.

4.5.1 Preliminaries

Primary context providers $p \in P$ are defined independent of the transaction-based processing model. This is due to the way how they acquire and provide contextual information, their operational behavior which can be described as autonomous and proactive (cf. Definition 4.3), and how they interact with the framework. The availability of a new context model m_p emitted by a primary context provider $p \in P$ causes the instantiation of a new instance of a context acquisition workflow $wf_{o_p} \in WF$ and initiates the invocation of the context acquisition activities of all compatible complementary context providers $c \in C_p$ orchestrated in the corresponding orchestration tree $o_p \in O$.

Every single activity carried out by a complementary context provider $c \in V_p$ to gather contextual information from the context source $s \in S$ it encapsulates is executed in a dedicated, self-contained, and independent *context acquisition process*[38] that controls and coordinates the operations appertaining to the acquisition, representation, and dissemination of contextual information. All acquisition activities defined by a context provider are encapsulated in one context acquisition process where the actual acquisition logic is defined external to the processing model. We use this term to distinguish it from context acquisition workflows (cf. Definition 4.8) which represent the sum of all context acquisition processes of the context providers $cp \in V_p$ orchestrated in an orchestration tree $o_p \in O$. We regard every single context acquisition process, denoted as $cap(c)$ in the following, as a *single atomic transaction* [GR92] that is completely decoupled and executed independently from other context acquisition processes. Context acquisition processes are instantiated and executed on demand after the context model m_{cp_i} of a

[37] The framework's name is deduced from the latin word '*acta*' meaning '*actions*'.

[38] For reasons for clarity and distinguishability, we denote the entire set of operations carried out by a complementary context provider $c \in C_p$ to create and emit a context model as *context acquisition process* in order to distinguish it from the set of all context acquisition processes pertaining to a context acquisition workflow $wf_{o_p} \in WF$.

preceding context provider $cp_i \in N^-_{o_p}(cp_j)$ was emitted. All the activities carried out by a context provider $cp \in V_p$ to transform the contextual information it acquires into a context model $m \in M_{cp}$, and to emit this context model for context augmentation and refinement are processed by the framework as one isolated transaction that either commits or aborts as a whole.

In this work, we therefore define a transaction as an atomic, self-contained, and autonomous process that involves all the necessary activities related to the acquisition, interpretation, clustering, transformation, consolidation, and dissemination of contextual information in form of an RDF-based context model that can be used by compatible context providers $c \in C_p$ for context refinement and augmentation. Technically, a transaction ensures that all operations involved in a context acquisition process of a context provider $p, c \in V_p$ are executed correctly and that only valid and consistent context models m emitted by $cp \in V_p$ are added to the compound context acquisition model $M_{o_p} \in OM$. In this respect, any context acquisition process $cap(cp)$ resulting in the dissemination of a context model $m \in M_{o_p}$ is regarded an *elementary* or *nested transaction* w.r.t. the context acquisition workflow $wf_{o_p} \in WP$ specified by an orchestration tree $o_p \in O$ it belongs to. As a corollary, a context acquisition workflow $wf_{o_p} \in WF$ that controls and monitors the context acquisition processes of the containing context providers $cp \in V_p$ and which is responsible for the aggregation of the emitted context models into a compound context acquisition model $M_{o_p} \in OM$ is regarded as one *global transaction* that consists of a distinct number of nested transactions that correspond to the elementary context acquisition processes $cap(c)$ of the context providers $c \in V_p$. In this respect, a context acquisition workflow $wf_{o_p} \in WF$ ensures that all nested transactions are executed consistently and that the context models $m \in M$ emitted by the orchestrated context providers $cp \in V_p$ are aggregated to a compound context model $M_{o_p} \in OM$ in a consistent and coherent manner.

To sustain this requirement, all nested transactions have a *temporal constant* assigned to them that determine a time frame in which a transaction must finish its acquisition activities and invoke a commit event, i.e., in which all the single operations pertaining to the acquisition of contextual information from a context source $s \in S$ must be completed. This constant is a necessary constituent to sustain a deterministic runtime behavior regarding the single context acquisition processes carried out within a context acquisition workflow $wf_{o_p} \in WF$, which would be difficult if transactions were conceded an arbitrary amount of time for the acquisition and dissemination of contextual information.

At any point in time, multiple context acquisition workflows $wf_{o_p} \in WF$ might be active while there is a 1:1:1-relationship between a primary context provider $p \in P$, the corresponding orchestration tree $o_p \in O$, and its context acquisition workflow $wf_p \in WF$. Every context acquisition workflow is considered an *atomic unit* where the containing context providers $cp \in V_{p_i}$ acquire their context models independently from context providers $cp \in V_{p_j}$ where $V_{p_i} \cap V_{p_j} = \emptyset$. These concepts resemble the idea of the *atomicity* and *isolation* properties defined in the *ACID paradigm* for database transactions [GR92]. In that sense, the context models $m \in M_{o_p}$ acquired within the course of their corresponding context acquisition processes are considered as atomic units where only the corresponding compound context model $M_{o_p} \in OM$ is updated as a whole for the creation of a new instance of the context configuration cc.

As a consequence, there exists exactly one distinct context acquisition workflow $wf_{o_p} \in WF$ for an orchestration tree $o_p \in O$ at any given point in time, i.e., one orchestration tree $o_p \in O$ can not be executed by two different context acquisition workflows $wf_i, wf_j \in WF$ and due to the transitive closure on the relations between the sets P, O, and WF, there does not exist two different global transactions for the same $o_p \in O$ for $i \neq j$. Furthermore, the set of all compound context acquisition models OM is regarded as the data basis from which the containing

compound context acquisition models $M_{o_p} \in OM$ are aggregated and transformed into the context configuration $cc \in CC$. The sequence of global transactions and thus the sequence of context acquisition workflows causes a context configuration to transition from a state q_i to a new state q_{i+1} that represents an updated manifestation of the user's current context.

The set of compound context acquisition models OM always contains exactly one concrete instance of a compound context model M_{o_p} pertaining to an orchestration tree $o_p \in O$. If a context acquisition workflow $wf_{o_p} \in WF$ also pertaining to $o_p \in O$ initiates the creation of a new compound context model M'_{o_p}[39], the previously created compound context acquisition model M_{o_p} which is already a member of OM is replaced by M'_{o_p} so that OM always contains only the most recently created compound context acquisition models. More precisely, this means that a compound context model $M_{o_{p_i}} \in OM$ created by a context acquisition workflow $wf_{o_{p_i}} \in WF$ at a time τ_i[40] is replaced by a compound context model $M'_{o_{p_i}} \in OM$ created by the same context acquisition workflow $wf_{o_{p_i}} \in WF$ at a time τ_{i+1}. Hence, the sets O, WF, P, and OM always contain the same amount of members where $|P| = |O| = |WF| = |OM|$.

One rationale of the processing model is that it employs a reactive behavior, i.e., the aggregation and consolidation process the result of which is a new context configuration instance $cc \in CC$ is only initiated, iff a new compound context acquisition model $M_{o_p} \in OM$ is available. This means that an updated version of the context configuration is not created unless a new compound context acquisition model $M_{o_p} \in OM$ was created and its pertaining global transaction was committed. Unless no new or updated contextual information has been acquired, the latest context configuration resembles a valid representation of the user's surrounding context.

For convenience reasons, we add the symbol of the context provider that emitted a context model as subscript to the symbol representing a context model (cf. Table 4.1). Hence, let m_{cp} denote a context model emitted by context provider $cp \in CP$. To add a temporal dimension to elements of the formal processing model, we use inverted commas as a complement to an element's symbol as this allows us to distinguish between different instances of the same element that have been created or modified at different points in time; for instance, let m'_{cp} denote the instance of a context model m pertaining to a context provider cp that has been emitted after m_{cp}.

In the following, we formally define the main constituting elements of the transaction-based processing model and present its main algorithms for the distributed acquisition and aggregation of contextual information. We base our definitions on elements defined by the ACTA formal framework and abridge or adapt them if necessary. Table 4.7 summarizes the symbols which have been used for defining the transaction-based processing model presented in this work.

4.5.2 Definition of Transactions

A fundamental element of the transactional processing model is the 1-to-1 correspondence that exists between the relations $r \in R_p$ representing the structure of an orchestration tree $o_p \in O$ and the transactions in which the context acquisition processes of the constituting complementary context providers $c \in C_p$ are executed. In this respect, the context acquisition workflow $wf_{o_p} \in WF$ of an orchestration tree $o_p \in O$ is considered as one global transaction that consists of a number of nested or elementary transactions (cf. Section 4.5.1), that is, all the context acquisition

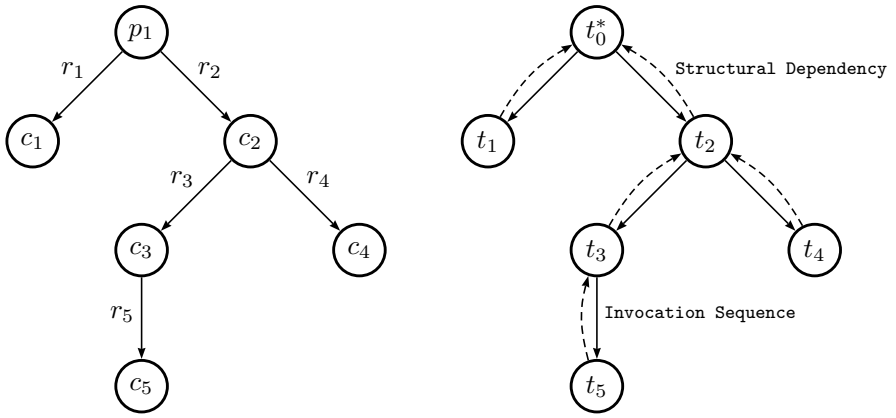
[39] For reasons of distinguishability, we denote the most recently created compound context acquisition model as M'_{o_p} in order to distinguish it from the previously created compound context acquisition model M_{o_p} which is already a member of OM.

[40] We use the symbol τ for designating a specific point in time to distinguish it from transactions which we denote as t.

Table 4.7: Symbols used in the transactional processing model

Symbol	Description
t	A nested or elementary transaction
T	Set of transactions $t \in T$
T_p	Distinct set of transactions $t \in T_p$ that correspond to the set of relations R_p of the orchestration tree o_p
G^{T_p}	Global transaction corresponding to an orchestration tree $o_p \in O$ and its context acquisition workflow $wf_{o_p} \in WF$ respectively
$cap(c)$	Context acquisition process of the complementary context provider $c \in C_p$ that executes all context acquisition activities defined by a context provider $c \in C_p$.
ob	Object that is created, modified, or deleted in the course of a transaction through an operation
op_t	Atomic operation executed in the course of a transaction $t \in T$
$op_t[ob]$	Operation op applied to an object ob in the course of a transaction $t \in T$
OP_t	Set of atomic operations $op \in OP_t$ that are defined for and can be executed by a transaction t
OE_t	Set of object events that can be invoked by a transaction t
SE_t	Set of significant events defined by the transaction model for the transactions $t \in T_p$
IE_t	Set of initiation events related to the invocation of a transaction $t \in T_p$
TE_t	Set of termination events related to the invocation of a transaction $t \in T_p$
$SE_{G^{T_p}}$	Set of significant events defined by the proposed transaction model that can only be invoked on global transactions G^{T_p}
$IE_{G^{T_p}}$	Set of initiation events that can only be invoked on global transactions G^{T_p}
$TE_{G^{T_p}}$	Set of termination events that can only be invoked on global transactions G^{T_p}
ϵ	Event created as a result of the execution of an operation op in the course of a transaction $t \in T$ on an object ob
$\epsilon \rightarrow \epsilon'$	Indicates that the invocation of an object- or significant event ϵ' is the result of the invocation of an event ϵ
τ	Temporal constant indicating the duration in which a transaction $t \in T_p$ must complete its acquisition operations, i.e., in which a termination event $\epsilon \in TE_t$ must be invoked
$','','''$	Inverted commas are used to indicate temporal aspects between element instances, e.g., a context model m was emitted previous to a context model m'
$N_T^+(t)$	Represents the set of transactions $t_{succ} \in T$ being adjacent to t such that t_{succ} succeeds t
$N_T^-(t)$	Represents the set of transactions $t_{prec} \in T$ being adjacent to t such that t_{prec} precedes t
H, H_t	Event history of both object events and significant events where the partition $H_t \subseteq H$ contains only those events invoked by a transaction t

processes $cap(c)$ of its constituting context providers $c \in c_p$ are regarded as nested transactions t that together constitute a global transaction. As a consequence, for every single relation $r \in R_p$ of an orchestration $o_p \in O$ there exists exactly one distinct transaction t. We equate the concepts of relations and transactions in this work, as a relation per definition (cf. Section 4.3) represents a context acquisition process between two compatible context providers $cp_i, cp_j \in V_p$ for $cp_i \neq cp_j$ that is initiated based on the existence of a context model m emitted by a preceding context provider $cp_i \in N_{o_p}^-(cp_j)$. Therefore, let the symbol t represent a transaction and be defined as an ordered list of atomic operations $op \in OP$, where OP is defined as the set of all atomic operations defined by the framework's processing model. Hence, a transaction t is an element from the set of all eligible combinations of atomic operations OP^n which are defined by the framework's internal acquisition model (cf. Table 4.3) and constitute the necessary elements for acquiring, transforming, and disseminating contextual information in form of a context model m. Formally, a transaction t can thus be defined as an n-tuple of operations $op \in OP$ as expressed



(a) Orchestration Tree $o_{p_1} \in O$ with five relations $r_{1\ldots5} \in R_{p_1}$ and six context providers $p, c \in V_{p_1}$

(b) Constituting structure of the global transaction $G^{T_{p_1}}$ corresponding to $o_{p_1} \in O$ depicted in (a)

FIGURE 4.9: Correspondence between the relations $r \in R_p$ constituting the structure of an orchestration tree $o_p \in O$ and the corresponding transactions $t \in T_p$

in Equation (4.55):

$$t := \langle op_0, op_1, \ldots, op_n \rangle \in OP^n \,. \tag{4.55}$$

Furthermore, let T be the general symbol for a set of transactions $t \in T$ where T_p specifically represents the set of transactions $t \in T_p$ that correspond to a concrete orchestration tree $o_p \in O$ respectively to the set of relations R_p that embody its structure. We explicitly express this correspondence through tuples (t_i, r_i) of a relation $R^{T_p,R_p}_{Corresponds}$ that is introduced as an addition to the relations defined in Table 4.2. The relation $R^{T_p,R_p}_{Corresponds}$ serves as a mean for formally expressing the correspondence that exists between relations $r \in R_p$ and transactions $t \in T_p$. As there corresponds per convention exactly one set T_p to one set of relations $R_p \in R(G)$ where $|R_p| = |T_p|$, let $R^{T_p,R_p}_{Corresponds}$ thus represent this 1-to-1 correspondence and be defined as the real subset of the Cartesian product of the sets T_p and R_p:

$$R^{T_p,R_p}_{Corresponds} \subset T_p \times R_p \tag{4.56}$$

Every relation $r \in R_p$ corresponds to exactly one transaction $t \in T_p$, that is, for every relation $r \in R_p$ there exists one distinct transaction $t \in T_p$ that corresponds to r as depicted in Figure 4.9. With Equation (4.56), this requirement is postulated in the following equation which we consider fundamental to the transaction-based processing model:

$$\exists!\, r \in R_p \,\wedge\, \exists!\, t \in T_p : (t, r) \in R^{T_p,R_p}_{Corresponds} \tag{4.57}$$

$$\forall r \in R_p \,\exists!\, t \in T_p : (t, r) \in R^{T_p,R_p}_{Corresponds} \tag{4.58}$$

As a consequence, Equation (4.56) and (4.57) introduce the necessary elements that allow us to define the set of transactions T_p on a formal basis:

$$T_p := \left\{ t \,|\, \exists\, r \in R_p : (t, r) \in R^{T_p,R_p}_{Corresponds} \right\} \tag{4.59}$$

As already expounded in Section 4.5.1, the contextual information emitted by the context

providers $p, c \in V_p$ as a result of the execution of their context acquisition processes are acquired and aggregated in the course of a global transaction. Therefore, let the symbol G^{T_p} represent a global transaction defined over the set of nested transactions T_p and formally expressed as the quadruple $G^{T_p} := \langle t_0^*, T_p, C_p, wf_{o_p} \rangle$ where t_0^* represents the initiating transaction, T_p represents the set of nested transactions t a global transaction consists of, C_p represents the set of complementary context provider $c \in C_p$ the context acquisition processes of which are executed by the nested transactions $t \in T_p$, and wf_{o_p} represents the context acquisition workflow through which a global transaction is executed.

The initiating transaction $t_0^* \notin T_p$ corresponds to the primary context provider an orchestration tree and context acquisition workflow pertains to and is defined as a special transaction since the acquisition behavior of a primary context provider is different from that of a complementary context provider and defined independent from the processing model primitives. By treating the acquisition and dissemination of a context model m_p by a primary context provider $p \in P$ as events that occurred in a special transaction t_0^*, the occurrence of the emit event can be used as an invocation of a global transaction G^{T_p}, if and only if a number of conditions are fulfilled (see Equation (4.75) and Algorithm 3).

The sequence of transactions $t \in T_p$ invoked by a global transaction G^{T_p} partially corresponds to the structure of the relations $r \in R_p$ of its pertaining orchestration tree $o_p \in O$, i.e., the structural composition of context providers $p, c \in C_p$ defined by the relations $r \in R_p$ of an orchestration tree $o_p \in O$ is per definition partially reflected in the invocation sequence of their corresponding transactions $t \in T_p$: if two relations $r_i, r_j \in R_p$ are adjacent to one another such that $\omega(r_i) = \alpha(r_j)$, the invocation order of their corresponding transactions always invokes $t_i \in T_p$ prior to $t_j \in T_p$, where t_j is only invoked when t_i successfully committed beforehand, otherwise t_j aborts (cf. Section 4.5.4). However, there exists only a partial correspondence between the composition structure of context providers $p, c \in V_p$ orchestrated in $o_p \in O$ and the invocation sequence of their corresponding transactions $t \in T_p$ as they are, depending on the structure encoded in the relations $r \in R_p$, invoked in parallel and operate independently and autonomously from one another. Thus, it is inherently impossible to determine their runtime behavior at design time. Referring to the orchestration tree illustrated in Figure 4.9, the transaction t_5 representing the context acquisition process of c_5 can invoke a commit event prior to c_1 although transaction t_1 had been started prior to t_5.

In general, there exists a bijective relationship between a transaction $t \in T_p$ and the context model m emitted by a context provider $c \in C_p$. The result of an instantiation of a transaction $t \in T_p$ is always the creation and dissemination of a context model m, if and only if t committed successfully. As a corollary, the number of transactions $t \in T_p$ that have invoked a commit event is always equal to the number of context models $m \in M_{o_p} \backslash m_p$ as the context model m_p emitted by the primary context provider $p \in V_p$ is processed external to the transactional model due to the operational behavior of p.

Any $wf_{o_p} \in WF$ that represents a running instance of the context acquisition workflow specified by an orchestration tree $o_p \in O$ is executed as a global transaction G^{T_p} where its sub transactions $t \in T_p$ are used to encapsulate and execute the context acquisition processes of the context providers $c \in C_p$ orchestrated in o_p. Let the relation $R^{G,O}_{Corresponds} \subset G \times O$ represent the correspondence that exists between the orchestration trees $o_p \in O$ and the global transactions G^{T_p} of their context acquisition workflows. Furthermore, each context acquisition workflow $wf_{o_p} \in WF$ and thus each global transaction G^{T_p} is executed in an isolated manner and is processed by the context framework as a single atomic unit that either commits or aborts as a

whole[41] (cf. [GR92]). In case of a commit or a partial commit (cf. Section 4.5.4), the pertaining compound context acquisition model M_{o_p} is added to the set of compound context models OM. Therefore, we postulate that set-theoretically there exists one distinct global transaction G^{T_p} for every $o_p \in O$ and any $wf_{o_p} \in WF$ respectively:

$$\forall o_p \in O \, \exists! \, G^{T_p} : (G^{T_p}, o_p) \in R^{G,O}_{Corresponds} \tag{4.60}$$

In addition, let $N^+_{T_p}(t)$ and $N^-_{T_p}(t)$ represent structural dependencies (cf. Section 4.5.4) that exist between the transactions $t \in T_p$ where $N^+_{T_p}(t)$ represents those transactions $t_{succ} \in T_p$ that are adjacent to $t \in T_p$ such that t_{succ} succeeds t, and $N^-_{T_p}(t)$ represents opposite transactions t_{prec} that precede t. In consequence, $N^+_{T_p}(t)$ represents those transactions $t_{succ} \in T_p$ that exhibit a structural dependency to t such that $(t_{succ} \, \mathscr{SD} \, t)$ holds (cf. Equation (4.64)) where a transaction t_{succ} can first begin when t_{succ} either commits or aborts. $N^+_{T_p}(t)$ is defined in Equation (4.61) for $i \neq j$ and $t \neq t_{succ}$:

$$\begin{aligned} N^+_{T_p}(t) := \Big\{ t_{succ} \in T_p \big| \, \exists r_i, r_j \in R_p : \omega(r_i) = \alpha(r_j) \\ \wedge \big((t, r_i) \in R^{T_p,R_p}_{Corresponds} \\ \wedge \, (t_{succ}, r_j) \in R^{T_p,R_p}_{Corresponds} \big) \Big\} \end{aligned} \tag{4.61}$$

In the same way, let $N^-_{T_p}(t)$ represent the opposite case and contain only those transactions t_{prec} that precede t such that the invocation of a begin event on t requires a commitment or abortion of t_{prec}. Equation (4.62) formally defines $N^-_{T_p}(t)$ for $i \neq j$ and $t \neq t_{prec}$:

$$\begin{aligned} N^-_{T_p}(t) := \Big\{ t_{prec} \in T_p \big| \, \exists r_i, r_j \in R_p : \omega(r_i) = \alpha(r_j) \\ \wedge \big((t_{prec}, r_i) \in R^{T_p,R_p}_{Corresponds} \\ \wedge \, (t, r_j) \in R^{T_p,R_p}_{Corresponds} \big) \Big\} \end{aligned} \tag{4.62}$$

4.5.3 Events and Event Histories

Analogous to the ACTA framework, we distinguish between events invoked by the execution of operations on objects, so-called *object events* and events that are caused by the invocation of transaction management primitives termed *significant events* (cf. [CR94]). An object event ϵ is the result of the invocation of an operation *op* on an object *ob* being specific to that object. The operations and hence the events are determined by the type of object an operation is executed on in the course of a transaction t. Thus, let the invocation[42] of an object event $\epsilon \in OE_t$ on an object $ob \in OB$ through the execution of an operation *op* by a transaction t expressed as $op_t[ob] \in OE_t$ where OE_t represents the set of object events "that can be invoked by t" [CR94].

Significant events, in contrast, are represented by the symbol SE_t or $SE_{G^{T_p}}$ and can be invoked on transactions defined in the transaction model. The subscript refers to the specific type of transaction a set of significant events is defined for. Furthermore, the set of significant events SE_t is compartmentalized into *initiation events* IE_t and *termination events* TE_t where $IE_t \cap TE_t = \emptyset$

[41] In addition to the usual termination events (commit and abort), we have defined a *partial commit* event (cf. Section 4.5.3) that allows a global transaction to commit even if not all of its nested transactions committed (cf. Equation (4.78)).

[42] We use the term 'invocation' analogous to its definition in the ACTA formalism, meaning to "cause an event to occur" [CR94].

and $IE_t \cup TE_t \subseteq SE_t$[43]. IE_t and TE_t define events that are related to the invocation and termination of a transaction $t \in T_p$ whereas $IE_{G^{T_p}}$ and $TE_{G^{T_p}}$ define equivalent events for global transactions G^{T_p}. Similar to the ACTA model, we perceive a transaction $t \in T$ as *in progress* if t has been invoked by an execution event defined in IE_t and has not yet been terminated by the execution of a termination event $\epsilon \in TE_t$.

For global transactions, we have extended the set of initiation events TE_t and introduce a special form of commit operation, which we call *partial commit*. A $\texttt{PartialCommit}_{G^{T_p}} \in TE_{G^{T_p}}$ refers to situations in which only a subset of the transactions t defined in T initiate a commit event $\texttt{Commit}_t \in H$. This is relevant for situations in which context sources $s \in S$ and, as a consequence, the context providers $cp \in CP$ that wrap those context sources become temporarily unavailable or deliver corrupted or incomplete data. In those cases, a global transaction G^{T_p} can invoke a $\texttt{PartialCommit}_{G^{T_p}} \in TE_{G^{T_p}}$ where only those context models $m \in M_{o_p}$ will be considered for the creation of a new instance of a context configuration $cc \in CC$ the corresponding transactions $t \in T_p$ of which have committed successfully. The rationale behind is that it is at least better to have incomplete information rather than to discard all context-relevant information acquired so far in the course of a context acquisition workflow $wf_{o_p} \in WF$. Whether or not a partial commit is invoked on a global transaction G^{T_p} depends on the current configuration of a framework's instance and might vary w.r.t. the current application scenario the framework is used in as in some situations it might be more useful to accept only completely aggregated context acquisition models.

In addition, we stipulate that every transaction $t \in T$ or global transaction G^{T_p} starts with a `Begin` and finishes with either a `Commit` or an `Abort` event. Global transactions G^{T_p} in addition might also finish with a $\texttt{PartialCommit}_{G^{T_p}}$ event. We define the following significant events:

- $SE_t := \{\texttt{Begin}, \texttt{Commit}, \texttt{Abort}\}$
- $IE_t := \{\texttt{Begin}\}$
- $TE_t := \{\texttt{Commit}, \texttt{Abort}\}$
- $SE_{G^{T_p}} := SE_t \cup \{\texttt{PartialCommit}\}$
- $IE_{G^{T_p}} := IE_t$
- $TE_{G^{T_p}} := TE_t \cup \{\texttt{PartialCommit}\}$

The effects of operations $op \in OP_t$ executed by a transaction $t \in T_p$ for creating a context model m_c are only made permanent when a $\texttt{Commit}_t \in TE_t$ event on the enclosing transaction $t \in T_p$ was invoked. As a consequence, the invocation of an emit event $\epsilon \in OE_t$ through the execution of the emit operation on a context model m_c by a transaction t with $emit_t[m_c] \in OE_t$ does not automatically cause the context model m_c to become a member of the set M_{o_p}. Instead, only the invocation of a $\texttt{Commit}_t \in TE_t$ event on the transaction $t \in T_p$ with $m_c \in M_{o_p} \Rightarrow \texttt{Commit}_t \in H$ can initiate the propagation of m_c and actuate its utilization by compatible context providers $c_{succ} \in N^+_{o_p}(c)$ for context aggregation, augmentation, and refinement in the course of the controlling context acquisition workflow $wf_{o_p} \in WF$. In consequence, the contextual information represented by the context models $m \in M_{o_p}$ can only be aggregated into a new instance of a context configuration $cc \in CC$ through the invocation of a $\texttt{Commit}_{G^{T_p}}$ event on the enclosing global transaction G^{T_p} (see Algorithm 7).

[43]Please note that these relationships also apply to significant events $SE_{G^{T_p}}$ defined for global transactions.

A fundamental concept in the ACTA formalism regarding the effects of concurrently executed transactions on other transactions as well as on objects is that of *histories* [BHG87, Bha99] that act as a mechanism to describe the correctness of the transaction management primitives of a transaction model through the properties of the history produced by a transaction model [CR94]. A history, denoted by the symbol H and defined for a set of concurrently executed transactions $t \in T$, thus "contains all events, significant events, and object events invoked by the transactions in T and indicate the (partial) order in which these events occur" [CR94][44].

To indicate temporal dependencies between transactions, ACTA defines the predicate $\epsilon \rightarrow \epsilon'$ as a complement to the definition of histories that allows to axiomatically express temporal relationships between events $\epsilon, \epsilon' \in H$; the predicate evaluates to true, if the occurrence of an event ϵ precedes the occurrence of event ϵ' in H. An event ϵ is a member of the history H and H_t if it was invoked by a transaction $t \in T_p$; in this respect, let ϵ_t denote the invocation of an event ϵ in a transaction t with $\epsilon \in H \Rightarrow \exists\, t \in T_p : \epsilon_t \in H$.

In addition, let $Condition_H$ denote a condition that must be fulfilled for ϵ to be a member of H. For $\epsilon \in H \Rightarrow Condition_H$, $Condition_H$ represents the necessary condition for ϵ to be in H; for $Condition_H \Rightarrow (\epsilon \in H)$, $Condition_H$ represents the sufficient condition for ϵ to be a member of H (see [CR94]).

Moreover, we use the '⇒'-symbol in compliance with the ACTA formalism to denote an *implication* between an event ϵ and the conditions that must be satisfied to initiate the invocation of ϵ, irrespectively whether ϵ is an event caused by the execution of an operation *op* on an object *ob* or a significant event invoked by the execution of a transaction model primitive operation.

4.5.4 Dependencies between Transactions

As expounded previously, one benefit of the ACTA formalism is—besides its capability to define dependencies in general—that it allows for the definition of constraints on dependencies between concurrently operating transactions in an axiomatic way while considering interaction and collaboration semantics. On the basis of the ACTA framework, we define the dependencies that are discussed in this section as being relevant for the transaction-based acquisition and dissemination of contextual information. However, we distinguish between the following types of dependencies:

- Structural or logical dependency that exists between transactions $t \in T_p$
- Commit dependency that exists between transactions $t \in T_p$ and G^{T_p}
- Abort and weak-abort dependency that exists between nested transactions $t \in T_p$ and global transactions G^{T_p}
- Begin and termination dependency for global transactions G^{T_p} and nested transactions $t \in T_p$
- Serial dependency that exists between nested transactions $t \in T_p$

In the following, we formally define the dependencies being relevant for the transaction-based acquisition and dissemination of contextual information; these definitions have been motivated by related works such as [CR91b, CR94, STS98]:

[44] Moreover, the ACTA formalism further distinguishes between a *complete history* denoted by the symbol H, and the *current history* represented by the symbol H_{ct}, and between projections of H called *subhistories* that satisfy a specific criterion defined for H.

- *Structural Dependency* ($t_j \mathscr{SD} t_i$): The structural composition of context providers $p, c \in V_p$ in an orchestration tree $o_p \in O$ expressed through the relations $r \in R_p$ can be directly mapped to the logical structure, i.e., the structural or logical dependencies that exist between the transactions $t \in T_p$. $\mathscr{SD}$ thus indicates a structural dependency between a pair of transactions $t_i, t_j \in T_p$ that is deduced from the composition of context providers $p, c \in V_p$ in an orchestration tree $o_p \in O$ and the relations $r \in R_p$ that define its structure. A structural dependency $(t_j \mathscr{SD} t_i)$ exists between two transactions $t_i, t_j \in T$ if and only if there exists an $r_i, r_j \in R_p$ such that $\omega(r_i) = \alpha(r_j)$ for $i \neq j$:

$$(t_j \mathscr{SD} t_i) \Rightarrow \exists\, r_i, r_j \in R_p : \omega(r_i) = \alpha(r_j) \tag{4.63}$$

  The logical structure of transactions $t \in T_p$ deduced from the structural composition of context providers $p, c \in V_p$ implies that a transaction t_j can neither invoke a $\texttt{Begin}_{t_j}$ event unless any other than its preceding transaction $t_i \in N^-_{T_p}(t_j)$ either commits or aborts, nor that any of the succeeding transactions $t_{succ} \in N^+_{T_p}(t_j)$ can invoke a $\texttt{Begin}_{t_{succ}}$ event prior to a termination of t_j, iff the corresponding relation r_i precedes r_j such that $r_i \in \delta^-_{o_p}\big(\alpha(r_j)\big)$. This means that if two relations $r_i, r_j \in R_p$ are adjacent to one another such that $\omega(r_i) = \alpha(r_j)$ for $i \neq j$ (cf. Equation (4.63)), the same holds for the pertaining transactions $t_i, t_j \in T_p$ with $(t_i, r_i) \in R^{T_p,R_p}_{Corresponds}$ and $(t_j, r_j) \in R^{T_p,R_p}_{Corresponds}$:

$$\begin{aligned} \exists\, t_i, t_j \in T_p, t_i \neq t_j, (t_j \mathscr{SD} t_i) \Longleftrightarrow \exists\, r_i, r_j \in R_p, r_i \neq r_j \\ \wedge\, (t_i, r_i) \in R^{T_p,R_p}_{Corresponds} \\ \wedge\, (t_j, r_j) \in R^{T_p,R_p}_{Corresponds} \end{aligned} \tag{4.64}$$

- *Commit Dependency* ($t_j \mathscr{CD} t_i$): A commit dependency exists between two transactions $t_i, t_j \in T_p$, if and only if there exists a *Structural Dependency* between $t_i, t_j \in T_p$ such that $(t_j \mathscr{SD} t_i)$, and both tuples (r_i, t_i) as well as (r_j, t_j) are members of the set $R^{T_p,R_p}_{Corresponds}$. This dependency is formally expressed in Equation (4.65):

$$\begin{aligned} \exists (t_j \mathscr{CD} t_i) \Longleftrightarrow \exists\, (t_j \mathscr{SD} t_i) : t_i, t_j \in T_p,\, t_i \neq t_j \\ \wedge\, (t_i, r_i) \in R^{T_p,R_p}_{Corresponds} \\ \wedge\, (t_j, r_j) \in R^{T_p,R_p}_{Corresponds} \end{aligned} \tag{4.65}$$

  A commit dependency between two transactions $t_i, t_j \in T_p$ implies that the invocation of a commit event $\texttt{Commit}_{t_j} \in H$ requires the precedent commitment of t_i, i.e., t_j can only commit, if and only if t_i committed before. With reference to the ACTA formal framework, this dependency is defined as follows[45]:

$$\texttt{Commit}_{t_j} \in H \Rightarrow \big(\texttt{Commit}_{t_i} \in H \Rightarrow (\texttt{Commit}_{t_i} \rightarrow \texttt{Commit}_{t_j})\big) \tag{4.66}$$

- *Begin Dependency* ($t \mathscr{BD} G^{T_p}$): A transaction $t \in T_p$ can only begin if the corresponding global transaction G^{T_p} has begun, i.e., the initiation of any nested transaction $t \in T_p$ requires the initiation of G^{T_p}:

$$\forall t \in T_p : \texttt{Begin}_t \in H \Rightarrow \texttt{Begin}_{G^{T_p}} \in H \wedge \big(\texttt{Begin}_{G^{T_p}} \rightarrow \texttt{Begin}_t\big) \tag{4.67}$$

- *Weak-Abort Dependency* ($t_j \mathscr{WAD} t_i$): A weak-abort dependency between two transactions $t_i, t_j \in T_p$ implies, that a commitment of transaction t_i precedes the abortion of a

[45] See Definition 2.3.1 on page 455.

transaction t_j in the history H (cf. [STS98]):

$$\texttt{Abort}_{t_j} \in H \Rightarrow \Big(\big((\texttt{Commit}_{t_i} \in H) \wedge (\texttt{Abort}_{t_j} \in H)\big) \Rightarrow \big(\texttt{Commit}_{t_i} \rightarrow \texttt{Abort}_{t_j}\big)\Big) \tag{4.68}$$

- *Abort Dependency* ($t_j \mathscr{AD}\, t_i$): An abort dependency ($t_j \mathscr{AD}\, t_i$) between two transactions $t_i, t_j \in T_p$ says that if t_i aborts so does t_j:

$$\texttt{Abort}_{t_j} \in H \Rightarrow \big((\texttt{Abort}_{t_i} \in H) \Rightarrow (\texttt{Abort}_{t_i} \rightarrow \texttt{Abort}_{t_j})\big) \tag{4.69}$$

  On the other hand, an abortion of t_j does not necessarily need to be caused by an abortion of t_i, i.e., t_i might commit successfully while t_j aborts (cf. *Weak-Abort Dependency*). As a consequence, if a transaction $t \in T_p$ aborts, any transaction $t_{succ} \in N^+_{T_p}(t)$ and subsequently following transactions also abort:

$$\forall t, t_{succ} \in T_p, t \neq t_{succ} : \texttt{Abort}_{t_{succ}} \in H \Rightarrow \texttt{Abort}_t \in H \tag{4.70}$$

- *Serial Dependency* ($t_j \mathscr{SD}\, t_i$)[46]: A transaction $t_j \in T_p$ can not begin until its preceding transaction $t_i \in N^-_{T_p}(t_j)$ either commits or aborts for $t_i \neq t_j$:

$$\texttt{Begin}_{t_j} \in H \Rightarrow \big((\texttt{Commit}_{t_i} \rightarrow \texttt{Begin}_{t_j}) \vee (\texttt{Abort}_{t_i} \rightarrow \texttt{Begin}_{t_j})\big) \tag{4.71}$$

In addition, a global transaction G^{T_p} can not initiate a termination event $\epsilon \in TE_{G^{T_p}}$ unless all of its nested transactions $t \in T_p$ have initiated a termination event $\epsilon_t \in TE_t$. This dependency is commonly designated as *Termination Dependency* ($G^{T_p}\, \mathscr{TD}\, T_p$) and exists between the nested transactions $t \in T_p$ and G^{T_p}, i.e., a global transaction G^{T_p} can neither invoke a commit nor abort event unless all of its nested transactions either commit or abort:

$$\big((\texttt{Commit}_{G^{T_p}} \in H) \vee (\texttt{Abort}_{G^{T_p}} \in H)\big) \Rightarrow \forall t \in T_p \big((\texttt{Commit}_t \in H) \vee (\texttt{Abort}_t \in H)\big) \tag{4.72}$$

More specifically, if all nested transactions $t \in T_p$ commit, then the corresponding global transaction G^{T_p} also commits; in the ACTA model, this dependency is denoted as a *Strong-Commit Dependency* [CR91b] and is formally defined in Equation (4.73):

$$\texttt{Commit}_{G^{T_p}} \in H \Rightarrow \forall t \in T_p \big((\texttt{Commit}_t \in H) \wedge (\texttt{Commit}_t \rightarrow \texttt{Commit}_{G^{T_p}})\big) \tag{4.73}$$

Two transactions $t_i, t_j \in T$ are considered *concurrent transactions* if their corresponding relations $r_i, r_j \in R_p$ with $(t_i, r_i) \in R^{T_p,R_p}_{Corresponds} \wedge (t_j, r_j) \in R^{T_p,R_p}_{Corresponds}$ fulfill the following condition: $\alpha(r_i) = \alpha(r_j)$ for $i \neq j$, that is, a context provider $p, c \in R_p$ is positive incident to both relations $r_i, r_j \in R_p$ and fulfills the previously stated condition. In consequence, $t_i, t_j \in T_p$ are executed concurrently if each of them exhibits a structural dependency to a preceding transaction $t_{prev} \in \big(N^-_{T_p}(t_i) \cap N^-_{T_p}(t_j)\big)$ where $N^-_{T_p}(t_i) = N^-_{T_p}(t_j)$, i.e., if there exists a structural dependency ($t_i\ \mathscr{SD}\ t_{prev}$) between t_{prev} and t_i as well as between t_{prev} and t_j with ($t_j\ \mathscr{CD}\ t_{prev}$). As expounded in Equation (4.65), two transactions $t_i, t_j \in T_p$ are *commit dependent* if a commit of t_j requires a commit of t_i for $i \neq j$; t_i, t_j are *independent* and thus processed concurrently if both t_i and t_j can commit or abort independently from each other.

In the following, we define the conditions that must be fulfilled to invoke a significant event on a nested transaction $t \in T_p$ or a global transaction G^{T_p} in an axiomatic way. For the following

[46]The Serial Dependency closely correlates with the Structural Dependency (cf. Equation (4.63) and (4.64)).

definitions, we introduce the shorthand notation m_t that refers to the context model m emitted by the context provider $\alpha(r)$ that is positive incident to the relation $r \in R_p$ which corresponds to the transaction $t \in T_p$ with $(t, r) \in R^{T_p,R_p}_{Corresponds}$. A transaction $t \in T$ and G^T terminates when either a $\texttt{Commit}_t \in TE_t$ or an $\texttt{Abort}_t \in TE_t$ event was invoked whereas a global transaction G^{T_p} might, depending on a framework's configuration, also invoke a $\texttt{PartialCommit}_{G^{T_p}}$ event (cf. Section 4.5.3). The conditions that must be fulfilled for either of the invocation and termination events differ between global transactions G^{T_p} and nested transactions $t \in T$ wherefore we define the corresponding axioms separately:

(1) A transaction $t \in T$ begins when its preceding transaction $t_{prec} \in N^-_{T_p}(t)$ has invoked a commit event and when the context model $m_{t_{prec}}$ emitted by the corresponding context provider $\alpha(r_{prec})$ with $\big(t_{prec}, r_{prec}\big) \in R^{T_p,R_p}_{Corresponds}$ is a member of M_{o_p}. Furthermore, the begin of a nested transaction $t \in T_p$ always requires the invocation of a $\texttt{Begin}_{G^{T_p}} \in H$ event of its corresponding global transaction G^{T_p}:

$$\texttt{Begin}_t \in H \Rightarrow \big((\texttt{Commit}_{t_{prec}} \in H) \wedge (\texttt{Begin}_{G^{T_p}} \in H) \wedge (m_{t_{prec}} \in M_{o_p})\big) \tag{4.74}$$

(2) A global transaction G^{T_p} initiates a begin $\texttt{Begin}_{G^{T_p}} \in IE_{G_{T_p}}$ if the corresponding primary context provider $p \in P$ detects a change in the context source $s \in S$ it encapsulates and emits an updated instance of a context model m_p:

$$\texttt{Begin}_{G^{T_p}} \in H \Rightarrow emit_{t^*_0}[m_p] \in H \tag{4.75}$$

(3) A nested transaction $t \in T$ commits, (i) if the context acquisition process of the pertaining complementary context provider $c \in C_p$ finished in a time $< \tau_{max}$, (ii) if an emit operation $emit_t[m_c] \in H$ was invoked on the context model m_c created as a result of the context acquisition process, and (iii) if m_c becomes a member of the set of compound context acquisition models M_{o_p}:

$$\texttt{Commit}_t \in H \Rightarrow \big((emit_t[m_c] \in H) \wedge (m_{cp} \in M_{o_p}) \wedge (\tau_{max} \geq \tau_{stop} - \tau_{start})\big) \tag{4.76}$$

(4) A global transaction G^{T_p} commits if all of its pertaining transactions $t \in T_p$ commit and if their commitment precedes the commitment of G^{T_p}:

$$\texttt{Commit}_{G^{T_p}} \in H \Rightarrow \forall t \in T_p : \big((\texttt{Commit}_t \in H) \wedge (\texttt{Commit}_\texttt{t} \rightarrow \texttt{Commit}_{G^{T_p}})\big) \tag{4.77}$$

(5) A global transaction G_{T_p} partially commits if there exists at least one nested transaction $t \in T_p$ that has invoked a $\texttt{Commit}_t \in H$ event and the context model m_t of which is a member of the set of compound context acquisition model M_{o_p}[47]:

$$\texttt{PartialCommit}_{G^{T_p}} \in H \Rightarrow \exists t \in T_p : \big((\texttt{Commit}_t \in H) \wedge (m_t \in M_{o_p})\big) \tag{4.78}$$

(6) A nested transaction $t \in T_p$ aborts in any of the following cases:

$$\texttt{Abort}_t \in H \Rightarrow \big(\neg(emit_t[m_c] \in H) \vee \neg(m_c \in M_{o_p}) \vee (\tau_{max} < \tau_{stop} - \tau_{start})\big) \tag{4.79}$$

[47] The constraints which transactions $t \in T$ have to commit in order to invoke a partial commit on a global transaction $\texttt{PartialCommit}_{G^{T_p}}$ can be defined individually among framework instances. Please note that this is a special case and overlaps with the conditions to invoke an $\texttt{Abort}_{G^{T_p}} \in TE_{G^{T_p}}$ event (cf. Equation (4.80)).

(7) A global transaction G^{T_p} aborts if at least one nested transaction $t \in T_p$ has invoked an abort event, i.e., if at least one nested transaction $t \in T_p$ has not invoked a commit event:

$$\texttt{Abort}_{G^{T_p}} \in H \Rightarrow \exists t \in T_p : (\texttt{Abort}_t \in H) \tag{4.80}$$

After having propound the necessary formal basis of a transaction-based processing model for the acquisition, dissemination, and aggregation of independently acquired contextual information while making full use of the ACID properties (cf. [GR92]), we now present an algorithmic description of the corresponding processing model and workflows.

4.5.5 Processing Context Acquisition Workflows

Algorithm 3 describes one of the elementary aspects of the context framework, that is, its reactive behavior related to the processing of available contextual information. The rationale behind is that the context framework autonomously and independently from any user action updates the semi-structured representation of the user's real world context in a proactive fashion by analyzing changes in terms of the acquired contextual information and initiating aggregation and consolidation tasks when appropriate.

Algorithm 3: Steps related to the initiation of a global transaction G^{T_p}

Precondition: Dissemination of a context model m_p by a primary context provider $p \in P$
Result: Invocation of a $\texttt{Begin}_{G^{T_p}} \in H$ event on G^{T_p} or rejection of context model m_p

```
1  if emit_{t0*}[m_p] ∈ H then
2      if (∄ wf_{o_p} ∈ WF : isRunning(wf_{o_p})) then
3          if (Commit_{G^{T_p}} ∉ H) ∧ (m_p ∉ M_{o_p}) then
4              initialize[M_{o_p}]
5              M_{o_p} ← (M_{o_p} ∪ m_p)
6              Begin[G^{T_p}]
7          else if ((Commit_{G^{T_p}} ∈ H) ∨ (m_p ∈ M_{o_p})) ∧ (∃ wf ≠ wf_{o_p} : isRunning(wf)) then
8              reset[M_{o_p}]                          /* Remove all m ∈ M_{o_p} from M_{o_p} */
9              replace[m_p]
10             Begin[G^{T_p}]
11         end
12     else
13         reject_{t0*}[m_p]
14     end
15 end
```

A new context acquisition workflow is initiated whenever a primary context provider $p \in P$ emits a new context model $m \in M_p$ indicated when the predicate $emit[m_p]$ becomes a member of H (line 1). As the availability of a new context model m_p usually initiates a new instance of a context acquisition workflow wf_{o_p}, the algorithm evaluates whether there exists an already running instance of a context acquisition workflow $wf_{o_p} \in WF$ pertaining to p and $o_p \in O$ respectively (line 2). This evaluation is mandatory because a primary context provider $p \in P$ might, due to its autonomous and self-contained behavior, emit a new context model $m'_p \in M_p$ immediately after $m_p \in M_p$ while the corresponding context acquisition workflow $wf_{o_p} \in WF$ and thus the global transaction G^{T_p} have not yet completed. If this is not the case and neither a $\texttt{Commit}_{G^{T_p}}$ event in H nor a context model $m_p \in M_{o_p}$ exists, the compound context acquisition model M_{o_p} pertaining to p is initialized and m_p becomes a member of M_{o_p}. As result, the

corresponding global transaction G^{T_p} is initiated (line 3-6). If a global transaction G^{T_p} pertaining to the context acquisition workflow $wf_{o_p} \in WF$ has previously invoked a $\texttt{Commit}_{G^{T_p}}$ event and there exists both, a complete compound context acquisition model $M_{o_p} \in OM$ as a result of the invocation of a commit event and other context acquisition workflows that run in parallel, the compound context acquisition model $M_{o_p} \in OM$ can be updated with the recently emitted context model m_p. Based on this event, a new context acquisition workflow wf_{o_p} is initiated to update the acquired contextual information. In consequence, the current instance of M_{o_p} is reset, i.e., all acquired context models $m \in M_{o_p}$ are removed and will be replaced by updated context models m'_c without violating consistency and completeness requirements (line 7-11). In any other cases, m_p is rejected so that a new context configuration creation process can be initiated[48].

Algorithm 4: Processing a context model m_t emitted by a transaction $t \in T_p$

Precondition: Existence of an emit operation $emit_t[m_t] \in \mathbb{M}$ in history H
Result: Invocation of a significant event $\epsilon_t \in SE_t$ on transaction $t \in T_p$

```
1  if (emit_t[m_t] ∈ H) ∧ (t ∈ T_p) ∧ (m_t ∈ 𝕄) then
2      if (∃ wf_{o_p} ∈ WF : isRunning(wf_{o_p})) ∧ (Begin_{G^{T_p}} ∈ H) then
3          if (m_t ∉ M_{o_p}) ∨ (Commit_t ∉ H) then
4              M_{o_p} ← (M_{o_p} ∪ m_t)
5              Commit[t]
6              foreach t_succ ∈ N+_{T_p}(t) : ((t_succ 𝒞𝒟 t) ∧ (t_succ 𝒮𝒟 t)) do
7                  Begin[t_succ]
8              end
9          else
10             reject[m_t]
11             Abort[t]
12         end
13     else
14         if (∄ wf_{o_p} ∈ WF : isRunning(wf_{o_p})) ∨ (Begin_{G^{T_p}} ∉ H) then
15             reject[m_t]
16             if Begin_t ∈ H then Abort[t]
17         end
18     end
19 else
20     if Abort_t ∈ H then
21         foreach t_succ ∈ N+_{T_p}(t) : (t_succ 𝒜𝒟 t) do
22             Abort_t[t_succ]
23         end
24     end
25 end
```

Algorithm 4 describes the workflow of processing a recently emitted context model $m_t \in \mathbb{M}$ by a complementary context provider $c \in C_p$ in the course of a transaction $t \in T_p$. If the $emit_t[m_t] \in H$ event invoked by a context provider $c \in C_p$ in transaction t pertains to a currently running instance of a context acquisition workflow $wf_{o_p} \in WF$ and if a $\texttt{Begin}_{G_{T_p}} \in H$ event on the pertaining global transaction G^{T_p} has already been invoked, then m_t becomes a new member of the compound context acquisition model M_{o_p} and t thus commits (line 1-5). If succeeding transactions $t_{succ} \in N^+_{T_p}(t)$ that exhibit a *Commit Dependency* as well as a *Serial Dependency* to t (cf. Equation (4.65) and (4.71)) do exist, those transactions can execute a $\texttt{Begin}[t_{succ}]$

[48] Depending on the configuration of a framework instance, an updated context model emitted by a primary context provider can also be buffered in the context description queue (see Section 4.6.2) and processed when the corresponding global transaction G^{T_p} has invoked a termination event.

operation and initiate their context acquisition processes (line 6-8). If m_t is already a member of M_{o_p} or the corresponding transaction has already invoked a $\texttt{Commit}_t$ event in the course of the same context acquisition workflow instance, then m_t is rejected and t aborts (line 10-11). In case there is no context acquisition workflow running where t pertains to, $m_t \in \mathbb{M}$ is rejected and an $\texttt{Abort}[t]$ operation is executed on t (line 14-16). Furthermore, in case an $\texttt{Abort}_t$ event on t is a member of the history H while one or more conditions specified in line 1 do not evaluate to true, all the succeeding transactions $t_{succ} \in N^+_{T_p}(t)$ that exhibit an *Abort Dependency* to t with $(t_{succ}\, \mathscr{AD}\, t)$ (cf. Equation (4.70)) also abort since succeeding transactions per Equation (4.65) and (4.71) can invoke a $\texttt{Commit}_{t_{succ}}$ event only if t commits (line 20-24).

Algorithm 5: Conditions for the invocation of a termination event on global transactions

Precondition: All transactions $t \in T_p$ have invoked a termination event $\epsilon \in TE_t$
Result: Invocation of a termination event on the enclosing global transaction G^{T_p}

```
1  if ∀t ∈ T_p : (∃ ε_t ∈ H ∧ ε_t ∈ TE_t) then
2  |  if ∀t ∈ T_p : Commit_t ∈ H then
3  |  |  Commit[G^{T_p}]
4  |  |  else if ∃t ∈ T_p : ((Commit_t ∈ H) ∧ (m_t ∈ M_{o_p})) then
5  |  |  |  PartialCommit[G^{T_p}]
6  |  |  |  else if ∄t ∈ T_p : (Commit_t ∈ H) then
7  |  |  |  |  Abort[G^{T_p}]
8  |  |  |  end
9  |  |  end
10 |  end
11 end
```

Algorithm 5 describes the workflow and conditions that must be fulfilled in order to invoke a termination event $\epsilon \in TE_{G^{T_p}}$ on a global transaction G^{T_p}. A termination event on a global transaction is invoked if a termination event has been invoked on all nested transactions $t \in T_p$ (cf. Equation (4.72)); Algorithm 5 evaluates this in line 1. In case all transactions $t \in T_p$ have invoked a $\texttt{Commit}_t$ event, a global transaction G^{T_p} also commits (cf. Equation (4.73)). With the introduction of a partial commit as an extension of the set of termination events $TE_{G^{T_p}}$, it is possible to define individual conditions that cause the invocation of a $\texttt{PartialCommit}_{G^{T_p}}$ event on a global transaction G^{T_p}. However, for reasons of simplicity we stipulated that at least one transaction $t \in T_p$ must have invoked a $\texttt{Commit}_t$ event in order to enable the invocation of a $\texttt{PartialCommit}_{G^{T_p}}$ event on the enclosing global transaction G^{T_p}. If all nested transactions $t \in T_p$ aborted, so does G^{T_p}.

The initiation of a global transaction G^{T_p} requires the invocation of a $\texttt{Begin}_{G^{T_p}} \in IE_{G^{T_p}}$ event on G^{T_p} and the availability of a context model m_p emitted by the corresponding primary context provider $p \in P$ where m_p has already been added as a member to the set of compound context acquisition models M_{o_p}. Algorithm 6 distinguishes between two cases where in the first case (line 2-6), there exist transactions $t \in T_p$ that exhibit a *Begin Dependency* (cf. Equation (4.67)) to a global transaction G^{T_p}. In this case, G^{T_p} consists of a set of nested transactions $t \in T_p$ wherefore their context acquisition processes need to be monitored by a context acquisition workflow wf_{o_p} being instantiated in line 3. Each nested transaction $t \in T_p$ that exhibit a *Begin Dependency* to G^{T_p} is then invoked through a $\texttt{Begin}_{G^{T_p}}$ event (line 4-6). If a global transaction G^{T_p} does not consist of a set of nested context acquisition processes, it can invoke a termination event instantly after its initiation and does not require an instantiation of a context acquisition workflow for monitoring the context acquisition processes of nested transactions (line 8-12). Depending on a positive or negative evaluation of the condition specified in line 8 which indirectly complies with Equation (4.72) and (4.73), a global transaction G^{T_p} either commits or aborts. Please note that

Algorithm 6: Steps performed by G^{T_p} after its invocation with $\texttt{Begin}_{G^{T_p}} \in IE_{G^{T_p}}$

Precondition: Existence of an invocation event on a global transaction G^{T_p} in history H
Result: Invocation of a termination event $\epsilon \in TE_{G^{T_p}}$ on a global transaction G^{T_p}

```
1  if ((Begin_{G^{T_p}} ∈ H) ∧ (m_p ∈ M_{o_p})) then
2  |  if ((T_p ≠ ∅) ∧ (∃ t ∈ T_p : (t 𝒟ℬ G^{T_p}))) then
3  |  |  instantiate_{G^{T_p}}[wf_{o_p}]
4  |  |  foreach t ∈ T_p : (t 𝒟ℬ G^{T_p}) do
5  |  |  |  Begin_{G^{T_p}}[t]
6  |  |  end
7  |  else
8  |  |  if (T_p = ∅) ∧ (∄ t ∈ T_p : (t 𝒟ℬ G^{T_p})) then
9  |  |  |  Commit[G^{T_p}]
10 |  |  else
11 |  |  |  Abort[G^{T_p}]
12 |  |  end
13 |  end
14 else
15 |  if (Begin_{G^{T_p}} ∈ H) ∧ ¬(Abort_{G^{T_p}} ∈ H) then
16 |  |  Abort[G^{T_p}]
17 |  end
18 end
```

a partial commit is not considered by Algorithm 4 as it can only be invoked in case there exist nested transactions $t \in T_p$. In case a $\texttt{Begin}_{G^{T_p}}$ event is contained in the history H but there does not exist a context model $m_p \in M_{o_p}$, G^{T_p} also aborts.

Algorithm 7: Instantiation of succeeding transactions $t_{succ} \in N^+_{T_p}(t)$

Precondition: Existence of a $\texttt{Commit}_t \in H$ event of transaction $t \in T_p$
Result: Invocation of an instantiation or termination event for transactions $t_{succ} \in N^+_{T_p}(t)$

```
1  if ((Commit_t ∈ H) ∧ (m_t ∈ M_{o_p}) ∧ (Begin_{G^{T_p}} ∈ H)) then
2  |  foreach t_succ ∈ N^+_{T_p}(t) do
3  |  |  Begin[t_succ]
4  |  end
5  else
6  |  if ((Abort_t ∈ H) ∨ ¬(m_t ∈ M_{o_p})) then
7  |  |  foreach t_succ ∈ N^+_{T_p}(t) do
8  |  |  |  Abort[t_succ]
9  |  |  end
10 |  end
11 end
```

The workflow and conditions that must be fulfilled for the invocation of a transaction $t \in T_p$ are described by Algorithm 7. The specified conditions comply with Equation (4.74), (4.76), and (4.79). As there exists a *Commit Dependency* (cf. Equation (4.67)) between a transaction t and its succeeding transactions $t_{succ} \in N^+_{T_p}(t)$, the inclusion of a $\texttt{Commit}_t$ event in history H together with the availability of a context model $m_t \in M_{o_p}$ emitted by the preceding context provider in t causes the invocation of a $\texttt{Begin}[t_{succ}]$ event on all succeeding transactions $t_{succ} \in N^+_{T_p}(t)$ that exhibit both a *Commit Dependency* and *Serial Dependency* to t (line 1-4). Based on Equation (4.79) an abortion of t causes all of its succeeding transactions $t_{succ} \in N^+_{T_p}(t)$ that exhibit an *Abort Dependency* to t (cf. Equation (4.70)) to abort too (line 6-10).

Algorithm 8: Operations related to the instantiation of a context acquisition process by $t \in T_p$

Data: Transaction t, Time frame τ
Result: Disseminating of context model m_t or invocation of an $\texttt{Abort}_t$ event

```
1  if ((Begin_t ∈ H) ∧ (m_{t_prec} ∈ M_{o_p})) then
2  |  acquire_t[m_{t_prec} ∈ M_{o_p}]
3  |  instantiate_t[cap(c_t)]
4  |  initialize_t[cap(c_t), m_{t_prec}]
5  |  initiate_t[cap(c_t)]
6  |  join_t[cap(c_t), τ]
7  |  if context model was acquired successfully and is ready for dissemination then
8  |  |  emit_t[m_t]                                  // Send context model to M_{o_p}
9  |  else
10 |  |  Abort[t]
11 |  end
12 end
```

When a $\texttt{Begin}_t$ event was invoked on a transaction t, the context acquisition process $cap(c)$ of the corresponding complementary context provider $c \in C_p$ is instantiated and initiated. Algorithm 8 describes the necessary operations being concerned with the acquisition and dissemination of contextual information gathered from a context source $s \in S$ wrapped by c. In a first step, the context model $m_{t_{prec}}$ emitted by its preceding context provider[49] is acquired from the set of compound context acquisition models M_{o_p} (line 2). In case this acquisition was successful, a new instance of a context acquisition process $cap(c_t)$ is created (line 3) and initialized by passing the context provider $c_t \in C_p$ pertaining to a transaction t together with the context model $m_{t_{prec}}$ of its preceding context provider as parameters to it (line 4). The initiation of a context acquisition process $cap(c_t)$ causes the context provider c_t to execute all of its operations relating to the acquisition of contextual information from the context source s it encapsulates. Usually this is performed in a decoupled and concurrent fashion where *cap* acts as a separate and self-contained runtime environment[50] that allows a context provider $c \in C_p$ to execute its context acquisition operations independently from a transaction t as well as concurrently running transactions pertaining to other context providers; the operation $join_t[cap(c_t), \tau]$ in line 6 represents this aspect, where a transaction t concedes a context provider c_t a particular time frame τ in which c_t must complete all of its acquisition operations $\in OP_t$. If all contextual information have been acquired successfully within the specified time frame, a transaction executes an $emit_t[m_t]$ operation signaling that a context model m_t is ready to be added as a member to the compound context acquisition model M_{o_p} (line 8). If a context provider c_t was not able to finish its acquisition operations within τ or became temporarily unavailable respectively malfunctioning during the acquisition process, transaction t initiates an $\texttt{Abort}$ operation on itself signaling that a context acquisition process could not be completed successfully (line 11). As outlined by Algorithm 7, the initiation of an $\texttt{Abort}$ operation on a transaction t causes all of its succeeding transactions $t_{succ} \in N^+_{T_p}(t)$ which exhibit both a *Serial Dependency* and *Abort Dependency* to t to abort, too.

If a primary context provider $p \in P$ that has already emitted its context model m_p so that $m_p \in M_{o_p}$ emits an updated version of its context model m'_p, its already stored context model $m_p \in M_{o_p}$ can be replaced by an updated version m'_p without violating consistency requirements if and only if other context acquisition workflows $wf \in WF$ are running in parallel and $C_p = R_p = \emptyset$. The same applies to primary context providers $p \in P$ with compatible complementary

[49] We presuppose that the preceding context provider acquired its context model $m_{t_{prec}}$ during the execution of transaction t_{prec} – see Equation (4.67) and (4.71).
[50] Usually this is realized using threads in a programing language.

Algorithm 9: Building a new instance of a context configuration $cc \in CC$

Precondition: There exist one or more context acquisition workflows that finished successfully between two consecutively following instances of context configuration $cc \in CC$

Result: Creation of a new instance of a context configuration $cc \in CC$

```
1  if (∄ wf_{o_p} ∈ WF : isRunning(wf_{o_p})) ∧ (∃ G^{T_p} : (Commit_{G^{T_p}} ∈ H ∨ PartialCommit_{G^{T_p}} ∈ H))
   then
2      acquire compound context acquisition models M_{o_p} ∈ OM      // collect all M_{o_p} ∈ OM
3      if ∃m ∈ M : ¬(isUpdated(m)) then
4          retrieve unaltered context models from M
5      end
6      cc ← f_merge(OM, Π)
7      send cc to replication manager
8      notify data provider dp ∈ DP
9      reset H, WF, OM
10 end
```

context providers $c \in C_p$ the context acquisition workflows $wf_{o_p} \in WF$ of which are already finished, that is, the emitted context models $m \in M_{o_p}$ are already aggregated to a compound context acquisition model M_{o_p}. In this case, a new global transaction G^{T_p} and thus a new instance of a context acquisition workflow wf_{o_p} will be instantiated and the context models $m \in M_{o_p}$ constituting the recently created compound context acquisition model M_{o_p} will be replaced by updated context models m'_{cp}.

Algorithm 9 describes the steps related to the building of a new instance of a context configuration $cc \in CC$. In case a context acquisition workflow $wf_{o_{p_i}} \in WF$ has finished, the context framework checks whether there exist further context acquisition workflows $wf_{o_{p_j}} \in WF$ where $i \neq j$ the corresponding global transactions of which have not invoked a termination event yet (line 1). If all acquisition workflows have finished, the acquired compound context acquisition models can be aggregated to a new instance of a context configuration cc by initiating the merging and consolidation process (cf. Equation (4.33)) in which all updated context models $m \in M$ as well as all unaltered context models are collected (line 2-6). Unaltered context models are those that pertain to compound context acquisition models the corresponding global transactions and context acquisition workflows respectively have not been initiated between two iterations of context configurations $cc_i, cc_j \in CC$ created at consecutively following points in time τ_i, τ_j where $\tau_j > \tau_i$ as their initiating primary context providers $p \in P$ have not observed any context updates in the context sources $s \in S$ they encapsulate. According to the conceptual system architecture proposed in Section 4.6, updated context models are retrieved from the *model manager* whereas unaltered context models are retrieved from the *context description queue*[51] which contains those context models that have been processed and aggregated by the context dispatcher in previous processing iterations. Both sets of context models are then aggregated where consolidation and reasoning rules are applied by a lightweight rule reasoner that is described in the conceptual architecture. After a new instance of the context configuration is built, it will be forwarded to the replication manager to initiate the replication tasks of the deployed data providers. Furthermore, the sets of context acquisition workflows WF and compound context acquisition models OM as well as the history H are reset, i.e., existing elements are removed (line 7-9).

[51]The context description queue is a data structure that hosts different versions of emitted context models and allows to revert to previously emitted context models for compensation purposes in case a context provider becomes temporarily unavailable and, hence, is unable to emit a context model.

4.6 Conceptual Architecture

4.6.1 Concepts and Features

For realizing the design considerations outlined at the beginning of this chapter, it is necessary to combine the processing of context information with the local replication of remote data sources. However, it is also necessary to keep the framework design as flexible as possible: it depends on the capabilities of the mobile device which context information can be tracked. Further, the user's information needs might evolve over time, hence the chosen approach cannot be restricted to a fixed set of remote data sources and should be flexible enough to enable the dynamic integration of new potential context sources on the fly. In the following, we highlight the main features exhibited by the conceptual architecture proposed in the current section in condensed form. These set of features are deduced from the requirements, design considerations, and the formal model discussed in the previous sections and serve as fundamentals of the proof-of-concept implementation being realized as a substantial part of the MobiSem project[52]:

- *Acquisition.* The acquisition architecture of the framework allows for acquiring context-relevant data from a wide variety of sources, ranging from locally deployed hardware and software sensors, over sensors located in ubiquitous environments, towards Web 2.0 APIs for retrieving data related to a user's personal or social networks. Raw sensor data acquired from such context sources are represented in form of RDF-based context description (cf. Section 4.4.1) and transformed into high-level context descriptions using concepts and languages from the Semantic Web.

- *Representation.* The architectural design of the context framework[53] has been drafted to impose minimal to none restrictions regarding the representation of contextual data and allows for using individual vocabularies and heuristics for explicitly representing context-related data semantics and reasoning on contextual data. Therefore, it emphasizes the use of standardized Semantic Web-based knowledge representation frameworks and languages such as RDF, RDFS, and OWL facilitating the reuse, exchange, and interoperability of context-relevant data.

- *Aggregation.* The capabilities to reuse, exchange, and augment contextual information to built richer and more elaborated context models are fundamental principles of the proposed framework and the underlying conceptual models. Context descriptions can be mutually refined and complemented with additional data in well-defined and controlled processes. Since context descriptions are represented as RDF graphs using standardized, well-known, and open Semantic Web vocabularies, their data semantics is understood across components.

- *Orchestration.* The implementation of an orchestration framework that analyzes the context providers' data descriptions and use them as a basis for cascading compatible context providers in orchestration workflows to dynamically route contextual information descriptions between them in order to refine and complement the contextual information space of the user is one of the main features of the framework. The orchestration framework thus

[52] MobiSem project website: `http://mobisem.org`

[53] We use the term 'context framework' as an abbreviated notion for the proposed context-sensitive RDF data replication framework as the main focus of our work lies on the acquisition, processing, and management of contextual information on mobile platforms (including its representation and interpretation) rather than on design and specification of specific replication heuristics or replication strategies.

has been designed to allow for the integration of individual orchestration rules and capability metrics for computing the compatibility matrix $\mathbf{M}^C$ and deriving the orchestration matrix $\mathbf{M}^O$. These matrices are dynamically and transparently re-created whenever a new context provider is integrated into the framework. We also decoupled the orchestration logic from the execution logic so that context providers can be orchestrated independently from their subsequent execution.

- *Consolidation.* To maintain context data consistency, data accurateness, and data completeness, context descriptions are collected and processed in a central place to guarantee consistency among context descriptions and acquisition processes while taking into account technical and operating system peculiarities of today's mobile platforms. Although there is no centralized control of context acquisition processes, contextual information is collected, aggregated, consolidated, and stored in a central component and repository. The feature of consolidation also covers the topic of *compensation management* as particularly in dynamic and mobile environments, the proper functionality of context providers as well as the sensors and services they wrap can never be universally guaranteed; it may happen that components become temporarily unavailable or that context providers emit incomplete or inaccurate data. Therefore, the context framework employs a set of compensation strategies to minimize the impact of such situations and to sustain a proper functionality of the context framework's processes.

- *Reasoning.* In order to consolidate the multitude of heterogeneous context information and to built a global, consistent, and coherent context model representing the user's current context, rule-based reasoning techniques are applied for context augmentation, aggregation, and the detection of context data inconsistencies. Since reasoning in general and on mobile platforms in particular is rather resource consuming, we employ a lightweight, forward-chaining rule reasoner where context-relevant reasoning rules are specified at design time. However, the reasoning component is currently in prototypical status and only allows for specifying hard-coded rules.

- *Dissemination.* User context is forwarded to other components deployed in the framework in an automated and transparent manner using a push-based notification mechanism. Additionally, external context services can request a model of the user's current context via an elementary HTTP server. The context framework exposes a communication infrastructure that allows other application or processes currently running on a mobile device to acquire context-relevant information via well-defined and transparent interfaces.

- *Replication.* Data is replicated in a transparent and automated fashion to the device where no restrictions are imposed upon the data sources nor a specific data representation scheme. The spectrum of potential data sources might range from files located on local or remote file systems, databases, web repositories, towards web applications and Linked Data sets [ZS11]. Data replication processes are completely decoupled from context acquisition processes, which is manifested in the absence of a direct connection between context and data providers. Mobile applications can access data replicas from a local triple store in a controlled and uniform way.

- *Storage.* The framework incorporates a persistence layer that allows for storing replicated data sets in a local database from where data replicas can be accessed and utilized by internal components. The persistence layer also includes support for *Named Graphs* [CBHS05] to address and distinguish among data replicas and contains projections for transforming RDF graphs into a relational database scheme and vice versa.

- *Data Access and Provision.* Access to replicated data stored in the local SQLite database is provided via an Android content provider that has been adapted for RDF data provision and storage. The RDF content provider assigns a unique URI to each replicated data sets and and offers external applications multi-granular access for utilizing the data contained in data replicas. It also incorporates an update and synchronization control mechanism that allows external applications to issue updates on data replicas without overwriting or compromising pristine data replicas.

The orchestration process can be configured to either perform a loose orchestration on the namespace level, or a detailed one by considering concepts and properties given by the context providers' data descriptions. When a new context provider is found in the system, the orchestration manager analyzes its data description and based on its configuration integrates the context provider in the orchestration graph $G(V, R)$. The compatibility score for each pair of context providers $cp_i, cp_j \in CP$ for $i \neq j$ is computed by a *matching algorithm* based on configurable scores for correspondences on the namespace, concept, and property levels. The algorithm for computing the compatibility score (cf. Section 4.4.2.3) for each pair of context providers $cp_i, cp_j \in CP$ for $i \neq j$ is built upon configurable scores for calculating the correspondences on the namespace, concept, and property levels. However, to make full use of the axiomatically defined semantics of RDF/RDFS and OWL elements, the arithmetic calculation of compatibility scores can be extended through the inclusion of RDFS semantics such as `rdfs:subClassOf` relationships. For instance, if one context provider $cp_i \in CP$ emits `foaf:Person` instances and another context provider $cp_j \in CP$ requires `foaf:Agent` instances as input data, the orchestration framework might be able to detect that compatibility between these differing concepts since `foaf:Person` is a subclass of `foaf:Agent` according to the FOAF ontology [BM07]. While running completely decoupled from other framework processes, rebalancing the orchestration graph $G(V, R)$ as a result of the orchestration process does not affect context acquisition tasks as such.

Figure 4.10 depicts a graphical illustration of the conceptual architecture and its main constituting components. The design principles of our proposed framework are based on the assumption of a homogeneous, locally deployed context-aware infrastructure where functional and technological compatibility is guaranteed by design. Context in general can be acquired *implicitly* and *explicitly* (cf. Chapter 2); the conceptual framework presented here allows for both forms of context acquisition although we focus on the implicit acquisition. In the following, we give an overview of each component of the conceptual system architecture of the proposed replication framework.

4.6.2 Components

4.6.2.1 Context Provider

A context provider captures a specific and relevant contextual aspect of the user's current context. A contextual aspect is represented in a structured and well-defined ways using semantic technologies (RDF, RDFS, OWL) to facilitate the exchange and integration of context information. The conceptual design of the replication framework allows to acquire contextual information from a wide variety of context sources and sensors. In this work, we refer to a sensor that emits context-relevant data as it was defined in [BC04]. Those sources are wrapped by context providers as expounded in Section 4.2, which employ two operation modes. Active context providers are primary and self-contained components that encapsulate a hardware or software

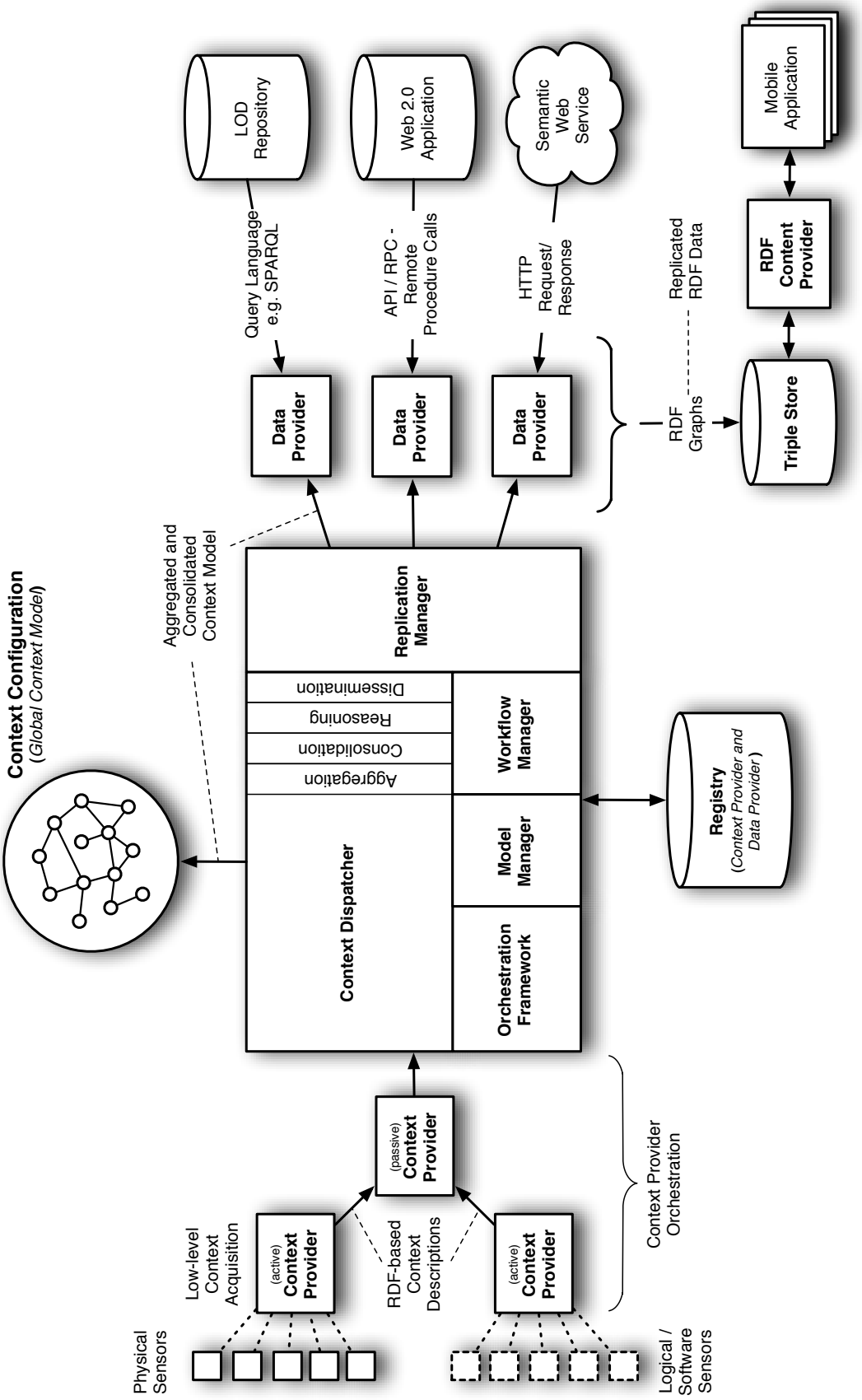


FIGURE 4.10: Conceptual architecture of the proposed context-dependent RDF data replication framework

sensor and operate independently and autonomously. They provide contextual information in a proactive manner and become active whenever a change in the corresponding context source is detected. Passive context providers are complementary and reactive components that react according to changes in primary context providers and become active when a corresponding primary context provider delivers an updated context model. They complement the contextual data retrieved from primary context providers by taking these context descriptions as input for initiating their acquisition tasks thus performing a form of *context augmentation*.

To provide the necessary flexibility in acquiring context-relevant data, context providers implement their own logic and heuristics for transforming any kind of input data (either sensorial or web-based content) into an RDF-based context description by using well-defined and well-accepted semantic vocabularies. As previously outlined, the acquisition of contextual data should not be restricted to capture sensorial data exclusively since the Internet and Web 2.0 applications in particular provide excellent sources for gathering context-relevant data. Context providers therefore are able to request data from four different types of sources:

(i) *Hardware sensors* that are integrated into the mobile system such as GPS module, luminosity sensor, camera etc. Most modern mobile platforms provide specific APIs for accessing and utilizing locally deployed hardware sensors capturing physical context (e.g., location, inclination, orientation, etc.).

(ii) *Ubiquitous sensors or devices* that are located in the physical environment [GSB02]. Such sensors must provide open accessible interfaces based on open network and access protocols and allow for acquiring mostly physical context.

(iii) *Software* or *logical sensors* wrap interfaces or APIs of local data sources mostly for complementing primarily acquired context information (e.g., Web services, APIs, online repositories, etc.) and allow for monitoring user or application behavior to deduce on the type of data that is relevant to the user in a specific situation.

(iv) *Web applications* such as Facebook[54], LinkedIn[55] etc. often contain useful information w.r.t the users' social relationships. Online and Linked Data repositories in particular provide magnitudes of freely available context-relevant data that can be exploited for complementing sensorially captured data.

By employing logical sensors, the acquisition of user-related contexts is emphasized. Such sensors can be adjusted towards a particular system infrastructure to gather context-relevant information by monitoring system processes to deduce information about the currently running applications as well as the data they operate on[56]. Context providers can make use of context descriptions from other context providers as well as external data sources; e.g., a component may use the GPS coordinates provided by another context provider to look up names of the current location using an external service[57]. A context provider describes the data it acquires but does not specify how these data are acquired.

[54] `http://developers.facebook.com/`

[55] `http://developer.linkedin.com/index.jspa`

[56] For the prototype implemented as a proof-of-concept, we developed software sensors that track user queries issued to various mobile applications such as browsers or the internal 'quicksearch'-function on an Android device.

[57] See Figure 5.5 in Section 5.2 for an example.

4.6.2.2 Context Dispatcher

The context dispatcher is the central component within the context framework. It handles the communication between context providers and propagates context models between them. Before propagating updated context descriptions to data provider components, the dispatcher performs additional processing on the data, like inference and consolidation (cf. Algorithm 9). Currently, the reasoning component uses (*i*) a generic lightweight rule-based reasoner, which allows to specify conditions under which new triples are added to the knowledge base, and (*ii*) hard-coded rules which are expressed by implementing a Java interface. The combination of these two mechanisms can, for instance, be used to specify that if one resource has multiple values for a functional property, the values denote the same resource (the corresponding rule (`A :ifp X`) $\wedge$ (`A :ifp Y`) $\Rightarrow$ (`X owl:sameAs Y`) can be interpreted by the rule-based reasoner), and that multiple resources that are related via a `owl:sameAs` property can be merged into a single resource in order to simplify further processing (a corresponding algorithm can be implemented as a Java class and integrated into the reasoning process).

Context descriptions are forwarded not only to data providers, but also back to context providers, so that they are enabled to mutually reuse and augment their context descriptions[58].

Communication between the context providers and the context dispatcher is realized via a context description queue that not only buffers the most recent context updates, but also stores previous revisions of context updates for compensation strategies in case a context source is temporarily not available or malfunctioning. In such cases, the context dispatcher can revert to previously committed context description to continue the context acquisition process. However, the context dispatcher employs some logic to maintain consistency among aggregated context descriptions.

4.6.2.3 Context Description Queue

Context providers communicate with the context dispatcher via the *context description queue*. The purpose of the context description queue is to buffer a context provider's recently emitted context descriptions depending on the status of the context dispatcher in order to resemble an asynchronous communication style as well as to serve as a means for implementing compensation management strategies. Whenever a context description together with the containing context model is sent to the context dispatcher, it is buffered in the context description queue until the context dispatcher is ready to process the context description and store the containing context model in the model manager. It implements specific logic for the management and exchange of context models to enable the recovery of lost descriptions and also serves as a *compensation repository* in case of temporarily unavailable context sources or malfunctioning context providers. Therefore, it not only stores different revisions of a context description but also implements a specific processing logic that allows the context framework to revert to previous context models in order to sustain the proper execution of context acquisition workflows and processes. In case a context provider becomes temporarily unavailable or malfunctioning and as a consequence is unable to emit a context model, the missing context model can be replaced by an earlier revision – although at the cost violating consistency requirement for reasons of context completeness (see Section 4.7). This is particularly relevant for orchestration trees in which multiple complementary context providers are orchestrated for context augmentation and refinement.

[58] Figure 5.5 depicts an example of augmenting GPS-coordinates with data from the `http://www.geonames.org` web service.

4.6.2.4 Model Manager

The model manager is used for storing and tracking the context providers' context models that have been acquired in the course of context acquisition workflow. For every context provider deployed in an instance of the replication framework, the model manager exhibits a specific data structure[59] in order to store the most recently emitted context model together with status information related to its acquisition activities. Whenever the context dispatcher successfully acquired a context description from the context description queue and processed the contained context model, it is stored in the model manager until a new instance of the context configuration is created. In this sense, the model manager acts as a both a controlling instance as well as a repository for context models processed by the context dispatcher.

4.6.2.5 Orchestration Framework

The orchestration framework handles all the tasks related to the computation of the compatibility scores and the creation of the corresponding orchestration trees being described in Section 4.4 and 4.4.3. It dynamically orchestrates compatible context providers by analyzing their data descriptions as described in the respective sections. The orchestration framework can be configured to either perform a loose orchestration on the namespace level, or a detailed one by considering concepts and properties given by the context providers' data descriptions. When a new context provider is found in the system, the orchestration framework analyzes its data description and based on its configuration integrates the context provider in a proper orchestration tree. While running completely decoupled from the context framework, rebalancing the orchestration graph does not affect context acquisition tasks as such.

The compatibility score for each pair of context providers is computed on the basis of configurable scores for indicating correspondences on the namespace, concept, and property levels. The orchestration algorithm performs an arithmetic matching based on data similarities and is additionally capable of including RDFS semantics such as `rdfs:subClassOf` relationships. For instance, if one context provider emits `foaf:Person` instances and another context provider requires `foaf:Agent` instances as input data, the matching algorithm detects the compatibility between these differing concepts since `foaf:Person` is a subclass of `foaf:Agent` according to the FOAF ontology [BM07].

4.6.2.6 Workflow Manager

The workflow manager is responsible for the management and coordination of the context acquisition workflows as well as the context providers' context acquisition processes that control and manage the acquisition activities of the context providers pertaining to an orchestration tree in order to sustain a deterministic and consistent behavior. Whenever a primary context provider emits a new context model, the workflow manager instantiates the corresponding global transaction and context acquisition workflow described in Algorithm 6 and controls the execution of the context acquisition processes. When all context acquisition workflows have finished, the workflow manager sets the corresponding entries in the model manager and notifies the context dispatcher that a new instance of the context configuration can be created.

[59] Technical details of the data structure implemented in the model manager together with further details are given in the MobiSem System Documentation [SZ10b].

4.6.2.7 Registry

The registry is the central storage and registration component where all context and data providers deployed in the framework must register in order to be integrated in acquisition and replication workflows. It uses separate hosting data structures and provides access to both primary and complementary context providers as well as data providers via dedicated methods and interfaces. It automatically notifies the orchestration framework whenever a new context provider has been registered so that it can be automatically orchestrated with compatible providers.

4.6.2.8 Context Configuration

The context configuration represents an aggregated version of all context providers' context models received by the context dispatcher. It is created when all context acquisition workflows being executed between the creation processes of two consecutive context configuration instances have been completed, i.e., when the corresponding global transactions have invoked a termination event (cf. Section 4.5.3) and the acquired context models are ready to be aggregated and consolidated. The context dispatcher then collects updated context models, aggregates them, applies reasoning rules as described before, and creates a new instance of the context configuration while maintaining context completeness, consistency, and accuracy.

4.6.2.9 Replication Manager

The replication manager controls and orchestrates all data replication tasks. It operates completely decoupled from the other framework components and gets notified by the context dispatcher whenever a new instance of a context configuration has been created. The replication manager is responsible for the instantiation of *data provider control threads*, which provide a runtime environment for the execution of the acquisition operations specified in a context acquisition process and encapsulating transaction; they control and monitor data replication tasks and propagate the context configuration to each data provider. The replication manager also receives notifications about changed data replicas in order to initiate write-back and synchronization operations.

4.6.2.10 Data Providers

Data providers are responsible for handling RDF data replication tasks; they replicate data of any kind to the mobile device and can request data from virtually any internal or external data source or generate data replicas themselves. Data provider components operate completely decoupled and independent from each other. Each data provider is assigned a named graph and unique identifier that is used as part of the addressing scheme it makes use of to store replicas in the local triple store. Data providers adjust and initiate their data replication tasks based on the analysis of the context configuration that they receive by the replication manager. For instance, a data provider may act upon changes of the current location and retrieve information about nearby points of interest.

In addition to the default data providers that merely retrieve data from remote sources and store them in the triple store, we have implemented a *selective checkout data provider* that makes use of a partial versioning mechanism for RDF triples based on triple bitmaps [Sch10] as well as a

write-back data provider that synchronizes the partially replicated data back to the repository, if the latter supports write operations (see Section 5.2).

4.6.2.11 Triple Store

Modern mobile platforms provide transparent access to persistent storage devices (e.g., flash memory cards) through a file system API. Therefore, the most straightforward way to store RDF data on a mobile device is to serialize it into a file on such a device using a standard RDF serialization format, like RDF/XML or N3. While this storage mechanism is extremely fast compared to DB-backed mobile storage solutions (cf. Chapter 6), it also has the significant disadvantage that RDF graphs must be completely loaded into the mobile device's working memory (RAM) before they can be further processed (e.g., before a SPARQL query can be issued). Alternatively, triples can be stored in a relational database, which causes an increase of read and write times but provides the possibility for structured queries over the data.

Our triple store implementation is designed to be a lightweight, efficient storage and retrieval mechanism for RDF triples. It abstracts over the concrete storage mechanism that is used by the mobile platform[60] and provides support for *named graphs* [CBHS05], persistence, and RDF serialization and de-serialization. It employs a *normalized table layout* (cf. [AMMH09]) where resources, literals, and blank nodes are stored in separate tables[61]. Regardless of which actual storage solution is used, it can be wrapped by a Java class that maps all read and write access methods to corresponding operations on the underlying physical representation (either flat files or a relational model). Currently, our triple store implementation does not perform in-memory buffering or caching. However, it can be wrapped by an additional in-memory `Graph` instance (which provides faster access) that regularly synchronizes itself with the database-backed instance.

4.6.2.12 RDF Content Provider

Applications can use RDF content provider to access data stored in the device's local triple store. The RDF content provider assigns to each replicated graph a unique URI, which can be used to access and retrieve the data contained in the graph. It exposes insert, update, delete and query methods and offers multi-grained access to data replicas, i.e., applications can access all replicas cached in the database, a specific replica, or a specific resource including all adhering triples of a specific replica.[62] In the background, the RDF content provider hides the details of context processing and data replication from applications; from the outside the replication framework looks like a common triple store whose data are regularly updated.

For our proof-of-concept implementation being described in more detail in [SZ10b], the RDF content provider is designed as a specialized Android Content Provider (see [RLMM09]) that is complemented with RDF processing and management capabilities to support the system-wide provision of RDF data replicas. It contains a number of projections for transforming RDF graphs into the relational database schema of the locally deployed SQLite database and vice versa. It

[60] Most mobile systems use specific storage systems such as the Record Management System (J2ME compatible devices) or a SQLite database (Google Android).

[61] A discussion regarding other database layouts for storing RDF triples including their advantages and limitations can be found in [HBS08].

[62] This functionality is implemented through an *Android Content Provider* that allows for defining explicit URI schemes for data replicas through which operating system-wide data access and data utilization is offered. By exposing distinct URIs (e.g. `content://org.mobisem.rdfprovider/graph#<graphid>`) triples can be retrieved, added, deleted, and updated.

exposes a common interface applications can use for performing query, update, insert, and delete operations on replicated data. The RDF content provider has been extended with named graphs support [CBHS05] and exposes configurable content URIs that allow for addressing specific parts of data replicas.

4.7 Discussion and Summary

After having defined our approach on a formal, algorithmic, and conceptual basis, we discuss a number of aspects and characteristics exhibited by the chosen approach that we consider essential for the aggregation and processing of contextual information.

- **Completeness of Contextual Information**

  The aspect of completeness is defined in relation to the context-relevant information space represented by the context sources $s \in S$. In general, it determines that all the contextual information acquired by the context providers $p, c \in V_p$ orchestrated in orchestration trees $o_p \in O$ should be aggregated into compound context acquisition models $M_{o_p} \in OM$ and be reflected in an instance of a context configuration $cc \in CC$. Completeness specifically defines that at any given point in time τ_i, an instance of a context configuration $cc_{\tau_i} \in CC$ contains the context models of all context providers $cp \in V_p$ orchestrated within the orchestration trees $o_p \in O$ deployed in a running instance of the framework. However, the aggregation algorithms, as outlined in Section 4.5, are defined on the principle of processing contextual information efficiently w.r.t. the available mobile operating system resources wherefore only the most recently acquired compound context acquisition models $M_{o_p} \in OM$ are integrated in the creation process of a context configuration $cc \in CC$. In combination with the reactive processing model, there might be situations in which only a designated number of context providers $p \in P$ have updated their context models between the creation of two subsequent instances of context configurations $cc_{\tau_i}, cc_{\tau_{i+1}} \in CC$. If only updated context information would be aggregated, an instance of a context configuration $cc \in CC$ would contain an incomplete representation of the user's context with respect to the entire set of primary context providers $p \in P$ and thus $cp \in V_p$.

  To illustrate this case, let P' be a partition of P and contain only those primary context providers that have emitted an updated context model between two subsequently created context configuration instances $cc_{\tau_i}, cc_{\tau_{i+1}} \in CC$ where $P' \cap (P \backslash P') = \emptyset$. If, as a consequence, only those compound context acquisition models M_{o_p} pertaining to the primary context providers $p \in P'$ would be included in the creation process of a new instance of a context configuration, $cc_{\tau_{i+1}}$ would contain an incomplete set of context information w.r.t. to entire context-relevant information space represented by the context sources $s \in S$.

  To circumvent this, an instance of a context configuration $cc \in CC$ hence contains the context models of all context providers $cp \in V_p$ orchestrated within the orchestration trees $o_p \in O$, even the context models of those context providers, which have not created an updated instance of their context models between the creation process of $cc_{\tau_i}, cc_{\tau_{i+1}} \in CC$ (cf. Algorithm 9). As a consequence, a context consumer thus always receives a complete representation of the user's current context including both the most recent context updates as well as infrequently changing context data acquired from previous context updates.

  In addition, the aspect of completeness is especially important in situations when during a context acquisition process, a context provider is malfunctioning or suddenly becomes

unavailable. In such situations, the framework is able to recover previously acquired context information from the context description queue and integrate those information into aggregation processes to maintain the aspect of completeness and to create a complete description of the user's current context.

- **Data and Processing Consistency**

  Consistency is a well-known *ACID property* for database transactions that guarantees a reliable, predictable, and deterministic execution of the operations pertaining to a transaction when those operations comply to the rules defined in a transaction model [GR92]. The proposed transaction-based processing model presented in Section 4.5 obeys to the consistency property and incorporates a consistency control on the basis of an extended transaction model to avoid inconsistencies on data and process level when processing and aggregating concurrently and distributedly acquired context information. It guarantees accurateness, trustworthiness, and reliability w.r.t. the data and activities involved in the context acquisition processes of context providers, which is considered a central aspect of distributedly operating context-aware systems (cf. [CEM03]). The transaction-based processing model facilitates and comprises two consistency dimensions:

  – *Data consistency* refers to the consistency of aggregated context-relevant data contained in a context model in terms of avoiding ambiguous, contradictory, incompatible, or outdated data as exemplified by the subsequently presented example.

  – *Processing consistency* refers to the fact that context acquisition processes and workflows are processed identically and independently from specific context provider instances regardless of temporal, contextual, technical, or external conditions.

  In general, a context model should not contain any outdated, unreliable, or contradictory data. The following example illustrates the emergence of data inconsistencies that might occur during the aggregation process of context models emitted by the context providers $cp \in V_p$ orchestrated in an orchestration tree $o_p \in O$. Let assume the existence of three context providers $p_1, c_2, c_3 \in V_{p_1}$ currently deployed in a running instance of the context framework that are sequentially orchestrated in a way such that c_2 receives a context model $m_{p_1} \in \mathbb{M}$ from p_1, and c_3 receives a context model $m_{c_2} \in \mathbb{M}$ from c_2. For this purpose, let assume that p_1 provides the geo-coordinates of the current position, c_2 transforms physical geo-coordinates into a location ID using a service such as *GeoNames.org*[63], and c_3 takes a location ID to query for points of interests (POIs) that are situated around a given location and represent them in its context model $m_{c_3} \in \mathbb{M}$.

  Let us further assume that p_1 acquires its context model $m_{p_1} \in \mathbb{M}$ at a certain point in time τ_i. c_2 thus receives m_{p_1} at time τ_{i+1} where $\tau_i < \tau_{i+1}$ and initiates its context acquisition process instantaneously. When c_2 has successfully updated its context model $m_{c_2} \in \mathbb{M}$ at a point in time τ_{i+2}, where $\tau_{i+2} > \tau_{i+1} > \tau_i$, it is forwarded to c_3 to retrieve POIs related to the current location ID.

  If p_1 creates and emits a new, updated context model $m'_{p_1} \in \mathbb{M}$ during the time c_2 is performing its acquisition activities, c_2 operates on the basis of the previous context model $m_{p_1} \in \mathbb{M}$, which is further used as input data for c_3. This situation would introduce a context model inconsistency as c_2 is performing its context acquisition tasks on the basis of the previously emitted context model m_{p_1} that contains superseded geo coordinates. As a consequence, a context configuration cc created by the context dispatcher would contain inconsistent data since the POIs retrieved by c_3 and the location ID created by c_2 do not

[63] GeoNames web service: `http://www.geonames.org/export/ws-overview.html`

correspond to the recently acquired geo-coordinates represented in the updated context model m'_{p_1} of p_1.

To avoid such data inconsistencies, the transaction-based processing model considers all context acquisition processes conducted within the course of a context acquisition workflow wf_{o_p} as constituents of a global transaction and therefore *locks* the corresponding primary context provider $p \in V_p$ as well as orchestrated complementary context providers $c \in C_p$ that have already emitted a context model so that no context updates can be propagated during the context acquisition processes of adjacent context providers $c_{succ} \in N^+_{o_p}(c)$ that are currently in progress. When a context acquisition workflow $wf_{o_p} \in WF$ has been executed successfully and the corresponding global transaction G^{T_p} has invoked a $\texttt{Commit}_{G^{T_p}} \in H$ event, the locks of all context providers $cp \in V_p$ are released so that new instances of emitted context models can be processed by the context dispatcher.

Consistency also concerns the synchronicity between different autonomous context acquisition workflows. Recent works therefore aim to apply the technique of *location fingerprints* [ACC09] where each context information fragment might be annotated with a unique location fingerprint to identify the location from where a certain contextual aspect has been acquired. Other related works apply a *temporal synchronization raster* for the acquisition of contextual data, where the time a context value is usable within the system is considered as a determining factor.

In addition to data consistency, consistency among context acquisition and aggregation processes is another important aspect – especially in unpredictable, indeterministic, and frequently changing situations. The replication framework must exhibit a consistent and deterministic behavior in case a failure occurs during a context acquisition process which causes a context provider to become temporarily unavailable or to lose connection to the context source it encapsulates. Moreover, if a specific behavior is repeated multiple times, the framework must behave identically and deterministically. Processing consistency also ensures that the same context-relevant information items are processed identically, i.e, the context acquisition processes of every context provider are executed and processed according to a consistent and coherent processing logic (cf. Section 4.3.5 and Section 4.5). It further ensures that all context acquisition workflows are executed and controlled identically regardless of specific context provider instances orchestrated in the corresponding orchestration trees. Consistency also concerns the orchestration processes in such way that for an identical set of data descriptions, identical compatibility scores and thus orchestration trees are calculated and created.

In addition, the inclusion of time frames that span a temporal scope in which a complementary context provider $c \in C_p$ must have completed its acquisition activities helps in maintaining a deterministic runtime behavior of context acquisition processes and enclosing workflows. If a context provider $c \in C_p$ is unable to finish its acquisition activities within its predefined time frame, the context dispatcher assumes that the context provider is malfunctioning or temporarily unavailable and, as a consequence, unable to emit a context model wherefore it aborts the corresponding transaction $t \in T_p$ (cf. Section 4.5).

- **Up-to-dateness of Context Representations**

  A context configuration instance should always represent the most accurate description of the user's current context and contain the latest acquired context information when requested by a context consumer. This implies that the most recent context updates are to be included in the creation process of a context configuration instance $cc \in CC$ regardless of a specific point in time. The proposed transaction-based processing model supports the

aspect of up-to-dateness through the concurrent execution of both context acquisition workflows $wf_{o_p} \in WF$ as well as context acquisition processes pertaining to such workflows in the form of transactions while considering consistency requirements and ACID properties. Furthermore, the process of creating a new instance of a context configuration is accelerated as only those compound context acquisition models $M_{o_p} \in OM$ are updated, the corresponding workflows $wf_{o_p} \in WF$ and global transactions G^{T_p} of which have been executed successfully during two subsequently created context configuration instances. However, to comply with consistency requirements, there is often a trade-off to be made between the aspects of up-to-dateness and data consistency since updated contextual information can only be incorporated into local information structures created and processed during the execution of context aggregation workflows when conflicts or contradictions between already existent and acquired data can be precluded (cf. data consistency example discussed in the previous aspect). This means that while a context acquisition workflow $wf_{o_p} \in WF$ is currently in progress, context updates acquired by containing context providers $cp \in V_p$ that have already emitted a context model and invoked a $\texttt{Commit}_t \in H$ event on the executing transaction are rejected (cf. Algorithm 4). However, the context description queue incorporates a mechanism that locally buffers such updates and allows the context dispatcher to process them when both the corresponding acquisition workflow has been finished and the pertaining global transaction has invoked a termination event $\epsilon \in TE_{G^{T_p}}$ (cf. Algorithm 5 and Section 4.6.2).

- **Reactiveness of Processing Context Updates**

  The aspect of reactiveness specifically determines how fast the context framework is able to react on context changes manifested in context model updates, even when there exist other concurrently running context acquisition processes and context acquisition workflows. It ensures that the time a context update is detected and the time it is acquired and processed by the context dispatcher is reduced to the necessary minimum. Reactiveness is related to the requirement of *Up-to-dateness* since a context configuration instance as global representation of the user's context should always contain the most recent context updates. In this respect, the formal model proposed in this chapter tries to minimize the amount of time that lies between the detection of a context update and its transformation into a context model, and the time needed for an integration into the creation process of a new context configuration instance. Since reactiveness and up-to-dateness are contrary to the requirement of data consistency, the implicitly incorporated lock mechanism reflected by the processing model accounts for both the minimization of aggregation and processing times while avoiding the inclusion and emergence of contradictions, inconsistencies, and inaccurate contextual data.

  For instance, if a primary context provider $p \in P$ detects a context information update and emits an updated context model $m'_p \in \mathbb{M}$ but the context acquisition processes of the complementary context providers $c \in C_p$ are still in progress, the recently emitted updated context model m'_p is rejected and buffered in the context description queue until all of the orchestrated context providers $c \in C_p$ have finished their acquisition activities (cf. Algorithm 3). The framework therefore treats a context acquisition workflow $wf_{o_p} \in WF$ as a single atomic transaction (cf. Section 4.5.1) and automatically locks the context acquisition processes $cap(c)$ of pertaining context providers $c \in C_p$ that have already emitted their context models[64] so that no conflicting context updates are propagated while acquisition workflows are in progress. Due to efficiency reasons, it is not necessary to lock the context acquisition processes of context providers pertaining to unconcerned and

[64] This concerns context providers, the corresponding transactions of which have invoked a $\texttt{Commit}_t \in H$ event.

concurrently running context acquisition workflows as the context providers orchestrated in the orchestration trees $o_p \in O$ are per default disjunct (cf. Equation (4.15)). However, the adaptive lock mechanism defined in the course of the processing model locks only the context acquisition processes of those context providers that pertain to the context acquisition workflow of their orchestration tree $o_p \in O$ being currently in progress in order to not influence the reactiveness of context acquisition processes pertaining to context providers orchestrated in other orchestration trees.

Reactiveness also accounts for the time interval that lies between the detection and processing of a context update, and its propagation to complementary context providers $c \in C_p$ via the context dispatcher. The faster the context dispatcher is able to react on context updates and propagate updated context models to compatible context providers, the more unlikely it is that a context update becomes superseded and obsoleted. Therefore, the processing model was designed to minimize reaction times while not violating consistency requirements.

- **Computational Effort and Memory Consumption**

  The computational expenses and memory consumption associated with processing of context acquisition processes and workflows should be adjusted to the available system resources of mobile information systems and platforms (cf. Section 7.2). Ideally, computational processes are self-adaptive and allow, e.g., to exclude complex reasoning algorithms from aggregation processes in case of limited processing power to keep response times acceptable. The computational effort for the proposed framework is relatively high due to the complex concurrency- and necessary consistency-control mechanisms needed for the execution of multiple simultaneously running context acquisition processes and workflows while avoiding the emergence of context data heterogeneities, inconsistencies, and contradictions. Moreover, transforming contextual information into RDF-based context models on the basis of Semantic Web-based description frameworks and reasoning engines consumes considerable amounts of memory capacity and computational power (see Chapter 6) as RDF is inherently a rather resource-intensive data format.

  To keep the processing load of a framework balanced, the inclusion of *feedback loops* in acquisition and computation-intensive processes is favorable in order to adjust processing capacities with respect to technical conditions and available system resources (cf. Section 7.2). For instance, complex and computationally expensive reasoning heuristics should not be performed in case of low memory capacity to keep the amount of inferred triples small. As evident from the evaluation of mobile RDF frameworks (cf. Chapter 6), current RDF processing frameworks are only capable of handling RDF models of comparably small triple sizes efficiently. Therefore, it is important to reach a trade-off between the size of a context model and the computational effort needed to create a model. On the other hand, in case of the availability of extensive computational resources, a framework might make use of a more complex and comprehensive *redundancy management* to minimize the impact of component malfunctions or acquisition failures. The framework presented in this chapter exhibits a relatively basic redundancy management that operates on the basis of the context descriptions being buffered in the context description queue and allows to revert to previously emitted context models in case of transmission failures or a loss of a complete context description to maintain the aspect of completeness (cf. Section 4.6.2.3).

In this chapter, we have presented the foundations of our approach manifested in form of a formal model of its main constituents, a formal and algorithmic description of the orchestration logic and the deduction of orchestration trees, a transaction-based processing model for contextual

information and context acquisition workflows, and a conceptual system architecture including implementation-relevant details when appropriate. At the beginning of this chapter, we provided a consolidated collection of requirements which we consider fundamental for a context-sensitive RDF data replication framework for mobile devices, and which were acquired in the course of an analysis of related works. Those requirements serve as design considerations of our approach.

An elementary aspect of our approach is the orchestration of compatible context providers in orchestration networks, which we denote orchestration trees, and which allow for a controlled acquisition and aggregation of compatible contextual information for context augmentation and context refinement. Those compatibilities are computed on the basis of data descriptions that specify input and output data required by a context provider to perform its acquisition tasks. By analyzing the context providers' data descriptions, we calculate several individual compatibility scores between pairs of context providers, where the aggregated score indicates the degree of compatibility that exists between two context providers in terms of the contextual data they both emit and possibly require. The final scores are recorded in a compatibility and orchestration matrix from which the orchestration trees are deduced.

In addition to the formal model, we also defined a transaction-based processing model for the acquisition, aggregation, and transformation of contextual information while preserving consistency, accurateness, and completeness requirements as well as taking into account mobile information system's peculiarities. We defined the processing model on the basis of the ACTA formalism that allows for a characterization of the nature of interactions and their semantics, the types of dependencies that exist between them, as well as the effects of transactions on involved transactional objects. The transactional processing model treats context acquisition workflows as global transactions and the context acquisition processes pertaining to the complementary context providers orchestrated in an orchestration tree as nested or elementary transactions. Its main feature is its reactive behavior as a new instance of a context configuration is created only in case a context provider detected an updated in the context source it observers and emits an updated context model as result. The processing model guarantees that the creation process of a context configuration instance does not interfere with existing acquisition and aggregation process and that consistency and completeness requirements are preserved.

As conclusion of this chapter, we presented a conceptual system architecture and discussed selected aspects of the proposed approach that concern the completeness of contextual information, data and process consistency, the accurateness of aggregated contextual information as well as the reactiveness of context acquisition workflows and the computational effort needed in processing and controlling context acquisition workflows. The proposed framework works independently from existing mobile applications and rather provide them with the necessary information. From the perspective of an application, the proposed framework looks like a normal triple store that is regularly updated with local or remote data from various sources while being easily integrable in existing systems and application infrastructures.

Chapter 5

Implementation and Case Study

"Context is key in the development of new services that will impact social inclusion for the emerging information society."

Coutaz et al., *Context is Key* (2005)

After having introduced the formal model and conceptual architecture of our approach in the previous chapter, we now present selected aspects regarding the implementation of a proof-of-concept prototype as well as a case study that is built upon the prototype in this chapter. We particularly focus on the underlying class structure of context and data providers that allows to integrate such components with a minimum of necessary adaptation work as most of the acquisition and replication-related behavior of context and data providers (cf. Section 4.3 and 4.5) is encapsulated in abstract super classes. This allows developers to focus on logic and implementation-specific details being relevant for the acquisition, transformation, and consolidation of contextual information while all processing- and management-related tasks are handled automatically by the framework. Section 5.2 and 5.3 thus present merely a small excerpt of the proof-of-concept prototype; extensive details regarding the implementation of the most relevant components as well as on the realization of the constituting algorithms and concepts are presented in [SZ10b, SZ10a, ZS12b]:

- In [SZ10b] we document the main concepts related to the acquisition, processing, management, and dissemination of contextual information together with the main algorithms for the (partial) replication of RDF graphs and their processing on the mobile device. This includes algorithms for the versioning, selection, partial replication, and back-writing of RDF graphs, which are also discussed in [Sch10].
- In [SZ10a] we give in-depth details about the implementation of the context framework. We specifically describe the entire system and its package infrastructure as well as aspects related to the orchestration of context providers, processing context models, controlling and managing context acquisition workflows, and the communication infrastructure implemented between the framework components. We also provide a practical guide on how to develop context and data providers and integrate them properly in the framework.
- In [ZS12b] we address the aspects of disseminating and utilizing contextual information in general and RDF data replicas in particular and also provide an efficient concept for

the processing of context descriptions by mapping context processing states to unique bit-based patterns. We further present implementation details of the RDF content provider and illustrate how mobile applications can access and retrieve RDF data replicas via the interfaces provided by the RDF content provider.

However, the framework is designed as infrastructure and can be adapted to a wide array of potential application scenarios. In the second part of this chapter, we therefore present a typical use case of a mobile knowledge worker that is constantly provided with information related to her current location as well as the people she is likely to meet in the near future. We have focused on a rather general use case centered around the proactive provision of location-based data as this type of context information is the most relevant for mobile users (cf. [BZD02, Bar03, RVW05, HSMY08, SVLO+11]).

5.1 Development Platform

We have chosen to implement the proposed framework on the Google Android platform[1] since the underlying operating system and application model provide substantial advantages compared to other mobile operating system architectures such as J2ME. Android itself is an open software stack and operating environment for running Java applications on the *Dalvik Virtual Machine* which is especially optimized for mobile environments. It includes a lightweight relational SQLite database that offers dedicated libraries for its utilization within application and services to store data persistently. Such data can be shared across applications in a controlled manner using Android's inter-process communication model. In contrast to the hard-wired application models of desktop operating systems, Android offers an *intend-based* application model that allows an application to specify a certain kind of functionality it requires for data processing where the operating system chooses the application that best matches. Android resembles some well-established software design patters such as the Model-View-Controller (MVC) pattern to separate application logic from user interface design and underlying data models. It provides access to the core system operating functions through standard APIs as well as a complete multitasking environment where each application is executed within its own thread, thus providing the possibility to implement background services, like a synchronization process that is automatically activated when the mobile device has online connectivity to its home network (e.g., by automatically establishing a VPN connection within a public wireless local area network). Access to core-system libraries and functions is offered via native APIs that can be used by both native and non-native applications. Android employs an equal, non-prioritized execution policy for native and non-native applications that are executed in the same runtime, and offers a complete multithreading environment where applications can place computationally expensive tasks in separate threads, e.g., to maintain user interface and application responsiveness [Mei10].

The Android operating system consists of a Linux-based system kernel, a middleware, key applications, and a set of API libraries for accessing native system functions and natively deployed hardware sensors [And10]. It was released in 2007 under the auspices of the Open Handset Alliance[2], a coalition of 79 technology and mobile companies[3] and has gained noticeable attention not only in the mobile software development industry but also in the mobile computing research community since it represents a new generation of mobile application development platform

[1] Official homepage of the Android platform: `http://developer.android.com`
[2] Open Handset Alliance: `http://www.openhandsetalliance.com/`
[3] see `http://www.openhandsetalliance.com/oha_faq.html`

and software development environment [RLMM09]. One of the key architectural features of the Android platform is the open communication infrastructure where application can reuse functionality and exchange data in a controlled and flexible way. Android includes a highly specialized virtual machine that was designed for low-powered handheld devices as those devices usually "*lag behind their desktop counterparts in memory and speed by eight to ten years*" [HKM+10]. In contrast to conventional Java virtual machines which use a stack-based architecture for data storage, the Dalvik VM is built upon a register-based architecture and transforms generated Java classes into a performance and memory-optimized Dalvik-specific file format (`.dex`-file) whose space compared to conventional `.jar` archives is reduced by the factor 2 [HKM+10].

As opposed to other conventional mobile development platforms, the Android operating consists of four main components, which are briefly introduced in the following as they embody essential constituents mandatory for the realization of crucial aspects of the conceptual architecture presented in Section 4.6:

- *Activities.* Activities are the building blocks of Android applications and correspond to a single UI screen. Activities are self-contained pieces of executable code and are either instantiated by the operating system via `Intents` or by the user [RLMM09].
- *Services.* Android services can be compared to services on desktop computers that execute a certain piece of code and run in the background. Services usually do not expose a user interface and their lifetime ends when the device is shut down. A typical example of a service is a music-player that remains active even when the user switches to another application. Persistent background tasks are usually executed as services.
- *Broadcast and Intent Receivers.* Broadcast receivers listen for system-wide events issued by the operating system or other applications and respond to those events. They allow applications or services to receive Intents by executing an Intent Receiver that processes the data or requests issued from other applications. Applications or services expose their functionality via Intent Receivers that specify which type of request the application or service can satisfy. The Android operating system automatically routes such requests to those components that can handle them most appropriately. Intents are a central concept for the inter-application communication and interaction.
- *Content Providers.* Content providers are used to share data between applications or services. They expose a standard URI-based interface that can be system-wide requested to receive data from a content provider. If an application queries for data, the operating system automatically selects the most appropriate content provider among all registered providers. Application and services can register their content provider using a specific URI such as `content://framework/rdf/context` where components can consume such data by issuing requests to an exposed URI. In case there are multiple content providers registered for the same URI, the system will choose the most appropriate one.

The application model underlying the Android platform allows for a loose coupling of mobile applications and constituting activities which can be dynamically added, removed, or substituted by other Activities depending on the user's needs. Together with the concepts of Intents and URIs, Android offers a flexible execution environment [RLMM09]. This environment provides the following opportunities regarding the deployment of a replication framework that exhibits necessary functionality for an efficient acquisition, management, storage, and dissemination of context information:

- *Ad-hoc discovery and exploitation of context sources* A discovery service can be deployed to constantly scan the environment for exploitable context sensors and ubiquitous devices in order to facilitate their integration into the acquisition space of the context framework.
- *Deployment of the context framework as part of the infrastructure* A context framework can be realized as a background service that is executed transparently and autonomously from the user while not affecting any of their tasks nor requiring explicit user attention.
- *Dissemination and provision of RDF data replicas via well-defined interfaces* The concept of content providers can be adapted to allow for the controlled and fine-grained provision of replicated data sets as well as for the provision and utilization of context data.
- *Non-disruptive user notification and interaction requests* Broadcast receivers can notify the user about relevant events in a non-disruptive manner while providing specialized configuration views in case explicit user inputs are necessary, e.g., for acquiring permissions related to the integration of recently discovered ubiquitous sensors.
- *Utilization and exploitation of locally deployed sensors and infrastructure functions* The open and uniform architecture of the Android operating system allows for a broad and system-wide utilization of locally deployed sensors on a mobile device where native APIs offer uniform and controlled access to those peripheral hardware.

The context framework is realized as a Java-based Android application and can either be deployed as an application or as a (background) service. It encompasses the entire context processing and management life cycle, i.e., context acquisition, representation, interpretation, aggregation, consolidation, context reasoning, context dissemination, context storage, and context provision and utilization. Each aspect is implemented in separate packages and expose dedicated interfaces for its utilization, adaptation, and extension. The framework in general is responsible for executing and maintaining context processing, context management, and context storage workflows in a controlled and deterministic manner.

5.2 Implementation of Context and Data Providers

In this section, be give a brief overview of the abstract classes and interfaces pertaining to context and data providers. They specify the necessary member variables and methods every context and data provider must implement for its proper integration into the context framework and the context acquisition and orchestration workflows, as well as for the replication workflows initiated by the replication manager.

All the basic data of context and data providers such as name, identifier, data description, and status information are held in the `AbstractBasicProvider` class and specified by the `IBasicProvider` interface as depicted in Figure 5.1. It represents the skeleton that allows every context and data provider to be integrated in the respective workflows. The `AbstractContextProvider` class implements methods regarding the communication with the context description queue and the context dispatcher as well as for the creation and management of the context providers' context models. The method declarations defined in the `IContextProvider` interface represent a minimum functionality a context provider must exhibit and are to be implemented or delegated to concrete context provider implementations. Primary context providers must be inherited from the abstract class `ActiveContextProvider` whereas complementary context providers are deduced from the abstract class `PassiveContextProvider`. Both classes are by itself subclasses of

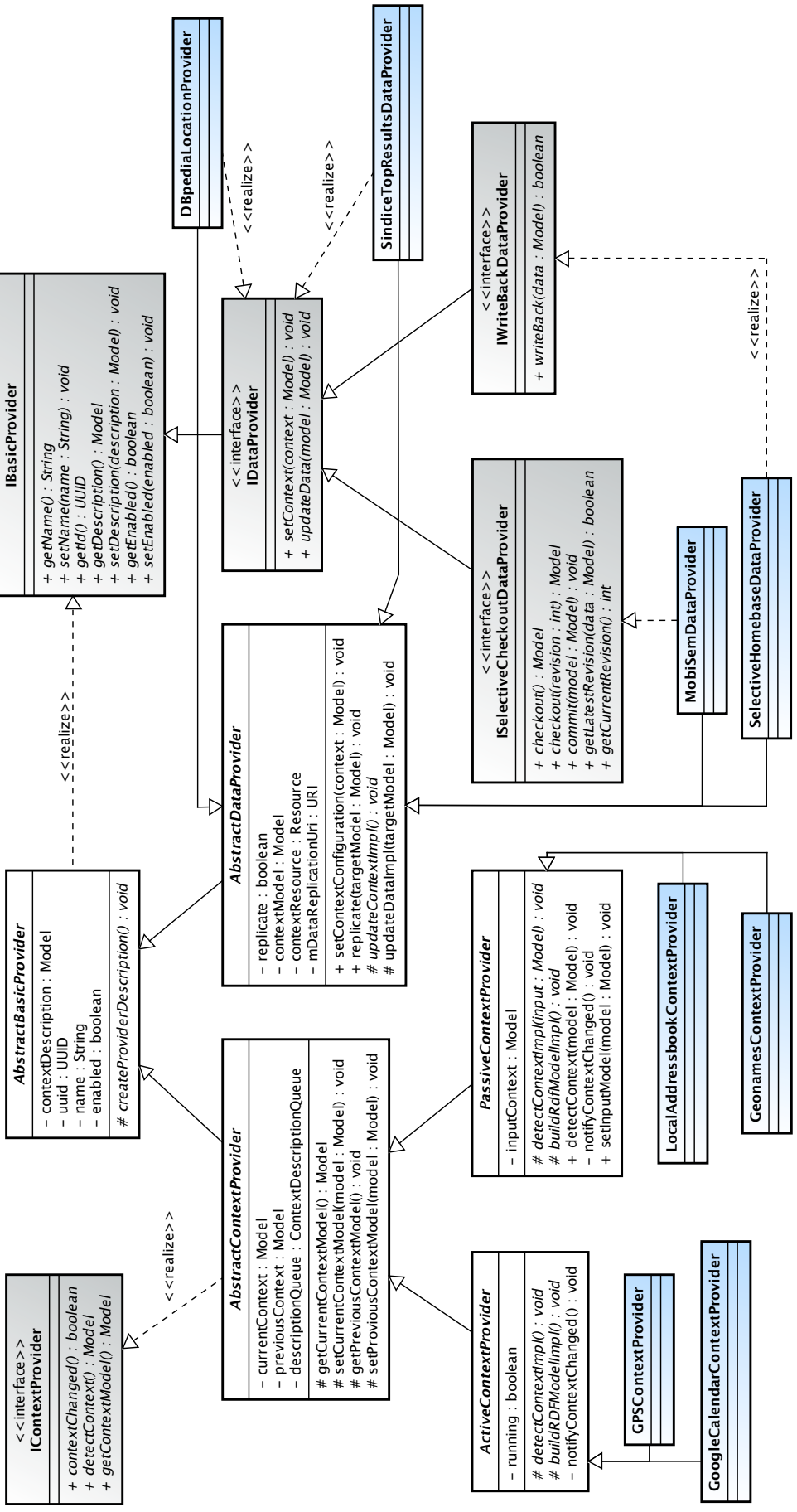


FIGURE 5.1: Conceptual class infrastructure for context and data providers

`AbstractContextProvider` and implement the *specific behavior* for each provider type. Moreover, they declare the abstract methods (i) `detectContextImpl()` in which the concrete acquisition logic is to be implemented, i.e., the methods and variables used to acquire context-relevant data from a context source, and (ii) `buildRdfModelImpl()` which holds the code for building the RDF-based context models emitted by a context provider. These abstract methods must be implemented in each concrete context provider individually and provides them with the opportunity to make use of individual acquisition techniques, communication protocols, interpretation and reasoning heuristics etc. As expounded in Section 4.2 and 4.6, primary context providers acquire context-relevant data independently from any request in a proactive and autonomous manner and, hence, run in a separate thread instance where their super class includes the infrastructure for their control. Complementary context providers initiate their context acquisition processes when they were notified by the `ContextDispatcher` about the availability of a required context input model emitted by a compatible context provider to which they are adjacent (cf. Equation (4.22) and (4.23)).

The generic functionality of data providers is implemented in the abstract class `AbstractDataProvider` that is a subclass of `AbstractBasicProvider`. It contains member variables for storing the context configuration and specifies the two abstract method declarations `updateContextImpl()` and `updateDataImpl()` that must be implemented by concrete data providers. The `updateContextImpl()` method allows a data provider to implement its individual processing logic and heuristics for analyzing the context configuration. The concrete replication code that is executed by a data provider after analyzing the context configuration must be implemented in the `updateDataImpl()` method, which is called by the replication manager to initiate a replication workflow. We decided to decouple the analyzation and replication process so that a data provider can perform additional processing before it initiates its replication workflow. The two interfaces `ISelectiveCheckoutDataProvider` and `IWriteBackDataProvider` contain method declarations for more complex data replication tasks, e.g., for replicating only subsets of data sets [Sch10] or bidirectional synchronization of replicated data.

For the case study, we implemented a number of concrete context providers, which are briefly presented in the following:

- `GPSContextProvider` is a primary (active) context provider that utilizes the device's internal GPS module for retrieving GPS coordinates. It can further be configured to use a WiFi-based location ascertainment method. The GPS context provider returns its data as latitude and longitude coordinates using the W3C WGS84 vocabulary[4].
- `GeonamesContextProvider` is a complementary (passive) context provider that is able to interpret geographical coordinates expressed using the W3C WGS84 vocabulary and requests the *GeoNames* web service[5] in order to resolve the GPS coordinates to a concrete geographical location identified by a GeoNames URI. Therefore, it is an ideal complement to the `GPSContextProvider` the output of which is enriched with a unique location ID being in addition to physical GPS coordinates that is also used in other data sets such as DBpedia (cf.[LBK+09]); however, this context provider is capable of processing arbitrary GPS coordinates independently from the source such coordinates were acquired.
- `GoogleCalendarContextProvider` is a primary (active) context provider that retrieves appointments from the user's calendar within a configurable time span (e.g., 72 hours) starting from the current time. For all events, a set of metadata are extracted and added to

[4]W3C WGS84 vocabulary: `http://www.w3.org/2003/01/geo/`
[5]GeoNames: `http://www.geonames.org/`

```
1  // handle location updates
2  // (called by Android location manager)
3  public void onLocationChanged(Location location) {
4     this.location = location;
5  }
6
7  @Override
8  // detect context model
9  // (called by ActiveContextProvider)
10 protected void detectContextImpl() {
11    // read data from the location object provided by a location sensor
12    double lat = this.location.getLatitude();
13    double lon = this.location.getLongitude();
14    String gpsProviderType = this.location.getProvider();
15
16
17    // clear context model
18    this.getCurrentContextModel().removeAll();
19
20    // create new context resources
21    Resource context = getCurrentContextModel().createResource();
22    context.addProperty(RDF.type, Vocabulary.Mobisem.CONTEXT);
23    Resource location = getCurrentContextModel().createResource();
24    context.addProperty(Vocabulary.Mobisem.CURRENT_LOCATION, location);
25
26    // attach coordinates to location resource
27    location.addProperty(Vocabulary.Geo.LAT, lat);
28    location.addProperty(Vocabulary.Geo.LONG, lon);
29    if (gpsProviderType != null)
30       location.addProperty(
31          Vocabulary.Mobisem.LOCATION_PROVIDER_TYPE, gpsProviderType);
32 }
```

FIGURE 5.2: Code snippet of `GPSContextProvider`, converting location data into an RDF-based context model

the context model, such as an event's name, start and end time, date, location descriptions associated to an event, and information about each event's attendees. It uses terms from the popular Dublin Core vocabulary[6] as well as the NEPOMUK calendar ontology[7] for modeling extracted metadata.

- `LocalAddressbookContextProvider` is a primary (active) context provider that exposes the contacts found in the user's local address book to make them available for relevant and interested data providers. Properties like name, email addresses, phone numbers, postal addresses, and instant messaging contact IDs are extracted and inserted into the context model, where they are expressed using the renowned FOAF vocabulary[8].

Typically, context providers convert their input data to an RDF model that represents this data in machine-processable form. Ideally, context providers use standardized, publicly available, and well-accepted vocabularies for this purpose in order to enable interoperability between communicating partners. As an example, Figure 5.2 shows how the `GPSContextProvider` builds a simple RDF model representing GPS coordinates obtained from the device's location sensor. In this example, the context provider registers itself as a location listener in the Android location subsystem; in consequence, the `onLocationChanged()` method is called every time the device's location

[6]Dublin Core vocabulary: `http://purl.org/dc/terms/`
[7]NEPOMUK calendar ontology: `http://www.semanticdesktop.org/ontologies/2007/04/02/ncal/`
[8]FOAF vocabulary: `http://xmlns.com/foaf/0.1/`

changes. In this case, the context provider buffers the most recent `Location` object. In turn, the `detectContextImpl()` method is regularly invoked by the `ActiveContextProvider` class. It creates a resource of type `mobisem:Context` and uses the property `mobisem:currentLocation` to associate the context with another resource, representing the actual location. This second resource is described with two properties for the latitude and longitude values, as well as one property for the type of the used location provider, if applicable.

Data providers are built upon the `AbstractBasicProvider` class and the `IBasicProvider` interface. Both components define methods and method declarations, and implement generic behavior that is common to both context and data providers. Data providers are responsible for handling all data replication tasks. They retrieve the context configuration from the replication manager which signals them to initiate their replication tasks based on their analysis of the relevant contextual aspects (cf. Definition 2.5) contained in the context configuration. Each data provider wraps a specific data source and replicates data to the local triple store built on top of the SQLite data base deployed on the Android platform. The framework does not impose any restrictions on the data sources to be wrapped by data providers; they can wrap a remote file system object, a remote data base, web services and web applications respectively, or generate data themselves. For instance, a data provider may act upon changes of the current location and retrieve information about nearby points of interest. Each data provider is assigned a named graph under which it stores its data in the triple store. The only requirement exposed is that replicated data must be available as RDF. The following concrete data providers were implemented for the case study:

- `DBpediaLocationDataProvider` is able to process geo coordinates as well as Geonames ids of location resources (which are taken from the context configuration). For all resources of type `geonames:Feature`, the data provider retrieves relevant location information from the DBpedia data set by querying the DBpedia SPARQL endpoint for relevant resources, based on their label. For all returned resources, descriptive triples are stored on the mobile device.

- `MobiSemDataProvider` is the default data provider for replicating data from a (versioned) MobiSem repository [SZ09a, SZ09b] to the mobile device. Through this client-/server-based infrastructure, any data source that can be expressed as RDF triples (e.g., file systems, relational data bases, etc.) can be exposed as versioned RDF graph repository, and can be replicated to a mobile device. For this purpose, a REST-style protocol has been developed that is used to specify additional metadata (like context information or the current graph version) in the form of additional HTTP headers. In addition to serving and versioning RDF graphs, the MobiSem repository is able to perform ranking and selection of RDF triples based on a context model that may be sent from the client alongside a checkout request. In this case, only parts of RDF graphs are replicated to the mobile device, saving transmission and storage capacity [SZ09a]. The `SelectiveHomebaseDataProvider`, in addition to providing support for such partial replicas, enables the client to write changes to replicated data back to the remote repository, which then takes care of merging and conflict detection [Sch10].

- `SindiceTopResultsDataProvider` extracts possible search terms from the context configuration by analyzing all literals found therein. It tokenizes the literals and constructs a list of most frequent keywords, which are in turn sent to the `Sindice.com` Semantic Web indexing service. From there it retrieves the top results and converts them into an RDF model, which is then stored on the mobile device. Optionally, the data provider is able to retrieve more information about the top results by de-referencing the resource URIs,

according to Linked Data principles (cf. [BL06a, BHBL08]). Any data retrieved via this option is also replicated and stored in the mobile device's triple store.

To illustrate the basic functionality of a data provider, we describe the relevant methods of the `DBpediaLocationDataProvider` in more detail (see Figure 5.3). It consists of two major methods: the first one, `updateContextImpl()`, is called by the data provider's implementation base class (`AbstractDataProvider`) whenever it receives an updated context model from the context dispatcher (cf. Figure 5.1). In this method, the data provider analyzes the context configuration passed as a μJena RDF model and iterates over all resources of type `geonames:Feature`; i.e., all logical locations. For all these resources, it extracts the labels and stores them in a list of resource labels. In the second method, `updateDataImpl()` (which is again called by the implementation base class whenever the data provider is requested to update data) a SPARQL `DESCRIBE` query is constructed that incorporates all resource labels (in this example we assume the user is interested only in information in English). Additionally it restricts the query to resources of type `dbpedia-owl:Place`. This query is sent to the DBpedia query endpoint, and the results are read into the `targetModel` variable, which represents a reference to the named graph used to store a data replica in the triple store.

Since DBpedia provides a large amount of information about locations (including names, descriptions, geographic and statistical data, as well as links to related persons, buildings, and events) such a data provider can be of great use in the case of our motivating example (cf. Section 1.2). The user will be enabled to browse relevant information about the places that she will visit during an upcoming trip without the need to actually search for it. By matching location information with a user's interests (which could be derived from the Web browsing history, or tags the user has used to annotate multimedia content such as photos) the system could recommend points of interests in a proactive way; by combining location data with publicly available personal profiles (e.g., from FOAF data) a user could be notified of people that live in cities the user is going to visit, and based on their interests it could suggest meetings with potential customers or cooperation partners.

5.3 Case Study

Referring to our motivating example (cf. Section 1.2), a component that provides location information can be used to update information while a user is on travel. It can be used for a variety of purposes, including the provisioning of general information about a location a person is visiting. Furthermore, it could be combined with information about personal interests in order to suggest points to visit in the user's spare time, or it could be combined with his calendar information in order to suggest public transport routes to the address where an event takes place.

To demonstrate the feasibility of our architecture, we have implemented a set of context and data providers on top of our prototypical framework implementation. The selection and design of these components is based on the assumption that the information needs of a mobile user depend on their current context (e.g., their location) as well as their future context. However, we want to emphasize that this framework is to be considered as an *infrastructure* upon which end-user applications that provide specific functionality, based on specific context information and replicated data, can be built.

```
// analyze the current context model (stored in this.contextModel)
// called when a new instance of the context configuration is propagated
// to the replication manager by the context dispatcher
@Override
protected void updateContextImpl() {
   this.currentResourceLabels = new ArrayList<String>();

   // iterate over all geonames features in the context model
   StmtIterator si1 = this.contextModel.listStatements(
      null, RDF.type, GEONAMES_Feature);
   while(si1.hasNext()) {
      Resource featureResource = si1.nextStatement().getSubject();

      // iterate over all statements of these features
      StmtIterator si2 = this.contextModel.listStatements(
         featureResource, null, (RDFNode) null);
      while(si2.hasNext()) {
         // check if a label property is attached
         Statement s = si2.nextStatement();
         if( s.getObject().isLiteral() &&
            ! this.currentResourceLabels.contains(s.getString())) {
            this.currentResourceLabels.add(s.getString());
         }
      }
   }
}

// update data from the remove data source
// called whenever a data provider is requested to replicate data
// from the remote data source
@Override
protected void updateDataImpl(Model targetModel) {
   // construct DESCRIBE query for all location resources
      StringBuffer queryBuffer = new StringBuffer();
   queryBuffer.append("DESCRIBE ?concept WHERE { \n");
   for(String featureLabel: this.currentResourceLabels) {
      queryBuffer.append("{" +
         "?concept rdfs:label \"" + featureLabel + "\"@en . " +
         "?concept rdf:type dbpedia-owl:Place . " +
      "} UNION \n");
   }
   queryBuffer.append("{} }");

   // send query to DBpedia
   String url = "http://dbpedia.org/sparql?query=" + URLEncoder.encode(
      queryBuffer.toString());
   HttpGet method = new HttpGet(url);
   method.addHeader("Accept", "text/plain");   // accept only N-TRIPLES
   try {
      // execute request
      HttpResponse response = new DefaultHttpClient().execute(method);
      // read model into targetModel for further processing by abstract superclass
      this.targetModel.read(response.getEntity().getContent(), "N-TRIPLE");
   } catch (Exception e) {
      // error handling
   }
}
```

FIGURE 5.3: Code snippet of the `DBpediaLocationDataProvider`, querying DBpedia for data about location resources

```
@prefix context: <http://www.mobisem.org/2009/01/context#> .
@prefix geo:     <http://www.w3.org/2003/01/geo/wgs84_pos#> .

[] a context:Context ;
   context:currentLocation [
      geo:lat "48.175443" ;
      geo:long "16.375493" .
   ] .
```

FIGURE 5.4: Geographical coordinates as returned by the `GPSContextProvider` (Turtle notation)

As outlined in Section 5.1, the implementation of the proof-of-concept prototype is based on the Android platform and uses the μJena Framework (cf. Section 3.3.2) to process RDF graphs[9]. In the following, we demonstrate how the proposed context-sensitive replication framework can be used to proactively provide RDF data from remotely operating data sources to the mobile device. Our objective is to permanently equip the user with data about the locations they are going to visit, about people they are likely to meet in the upcoming days, as well as people that are based near the user's current position. To accomplish this, different kinds of contextual information are utilized, including the device's current position and the user's calendar data.

5.3.1 Context Acquisition

For conducting the case study, we have implemented three distinct context providers on top of the proof-of-concept implementation of the framework: first, a primary location context provider (`GPSContextProvider`) uses the device's built-in GPS sensor to track geographical coordinates. It returns them in form of context descriptions that contain a `context:currentLocation` property (cf. Figure 5.4) to describe the coordinates of the current location using the WGS84 Geo Positioning vocabulary[10] that is built upon the World Geodetic System Scheme 84[11].

Second, a complementary context provider (`GeonamesContextProvider`) uses the GeoNames service[12] to resolve GPS coordinates to geographical entities. This component receives context updates from the context dispatcher, extracts properties that represent geographical coordinates, and returns information retrieved from the web service—in our example, a reference to a geographical entity as well as its name (cf. Figure 5.5).

In parallel, a third primary context provider (`GoogleCalendarContextProvider`) regularly scans the user's calendar and extracts all appointments within the next 72 hours. From these appointments the e-mail addresses of all participants are extracted and returned, as depicted in Figure 5.6 (in this case, two e-mail addresses are returned). Further, the locations of appointments are extracted and are returned as GeoNames features. This context provider uses terms from the NEPOMUK ontologies[13] and from FOAF to describe the extracted resources.

The context dispatcher—which receives notifications and updated instances of the context models emitted by the context providers every time a context property value changes—buffers, combines, and enriches the context models with additional information in case a new instance of the

[9] As shown in Chapter 6, μJena exposes a very weak performance compared to other RDF frameworks; however, more efficient implementations have been made available only recently. We plan to port our proof-of-concept implementation towards a more efficient RDF framework in the near future.

[10] WGS84 Geo Positioning: an RDF vocabulary: `http://www.w3.org/2003/01/geo/wgs84_pos#`

[11] World Geodetic System 1984 (WGS 84): `http://earth-info.nga.mil/GandG/wgs84/index.html`

[12] Geonames web service: `http://www.geonames.org`

[13] Overview of the NEPOMUK ontologies: `http://www.semanticdesktop.org/ontologies`

```
@prefix context:  <http://www.mobisem.org/2009/01/context#> .
@prefix geonames: <http://www.geonames.org/ontology#> .
@prefix rdfs:     <http://www.w3.org/2000/01/rdf-schema#> .

[] a context:Context ;
   context:currentLocation [
      geonames:nearby <http://sws.geonames.org/2761369/> .
   ] .
<http://sws.geonames.org/2761369/> a geonames:Feature ;
   rdfs:label "Vienna" .
```

FIGURE 5.5: Context description as returned by the `GeonamesContextProvider` (Turtle notation)

```
@prefix context:  <http://www.mobisem.org/2009/01/context#> .
@prefix ncal:     <http://www.semanticdesktop.org/ontologies/2007/04/02/ncal#> .
@prefix foaf:     <http://xmlns.com/foaf/0.1/> .
@prefix geonames: <http://www.geonames.org/ontology#> .
@prefix rdfs:     <http://www.w3.org/2000/01/rdf-schema#> .

[] a context:Context ;
   context:upcomingEvent [ ncal:attendee
         [ foaf:mbox <mailto:bernhard.schandl@univie.ac.at> ] ,
         [ foaf:mbox <mailto:stefan.zander@univie.ac.at> ] ;
      ncal:location [ a geonames:Feature ; rdfs:label "Munich" ] .
   ] .
```

FIGURE 5.6: Context retrieved from the user's calendar by `GoogleCalendarContextProvider`

context configuration is to be created (cf. Section 4.3 and 4.5). It merges all resources typed as `context:Context` into a single RDF model serving as new instance of a context configuration, assigns it a unique URI so that it can be referenced by other components or context models, and adds a timestamp as well as a link to the preceding context configuration instance. Moreover, it applies simple inference rules to the acquired and aggregated context models: for example, the `context:currentLocation` property has been defined as *functional property* (since we assume that the user can be at only one location at the same time), from which the lightweight rule-based reasoner (cf. Section 4.6.2) can deduce that the two anonymous location resources returned by the different context providers are actually identical respectively refer to the same location and can likewise be merged, as shown in Figure 5.7.

The context dispatcher propagates this new instance of the context configuration via the replication manager to all data providers deployed in the system whenever a contextual change has been acquired successfully by a context acquisition workflow (cf. Algorithm 9). It is then up to each data provider to decide whether to initiate a new replication task, and which information from the current context configuration instance they use for this purpose.

5.3.2 Data Provisioning

To simulate data sets being replicated in a real-world setting, we have implemented a number of data providers that address different information needs of mobile users and replicate data from different Linked Data sources to the mobile device. The `SindiceTopResultsDataProvider` uses the Sindice Semantic Web index[14] to retrieve information from FOAF descriptions being

[14]Sindice Semantic Web search engine: `http://sindice.com`

```
@prefix context:  <http://www.mobisem.org/2009/01/context#> .
@prefix geo:      <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix ncal:     <http://www.semanticdesktop.org/ontologies/2007/04/02/ncal#> .
@prefix foaf:     <http://xmlns.com/foaf/0.1/> .
@prefix geonames: <http://www.geonames.org/ontology#> .
@prefix rdfs:     <http://www.w3.org/2000/01/rdf-schema#> .

<urn:uuid:baac630a-5cdb-4c79-92e6-6ce3d07419bc>
   a context:Context ;
   context:timestamp "2009-06-16T15:58:22"^^xsd:dateTime ;
   context:previous <urn:uuid:d3ee316b-5704-4893-acb9-df1495c79011> ;
   context:currentLocation [
      geo:lat "48.175443" ;
      geo:long "16.375493" ;
      geonames:nearby <http://sws.geonames.org/2761369/> .
   ] ;
   context:upcomingEvent [ ncal:attendee
         [ foaf:mbox <mailto:bernhard.schandl@univie.ac.at> ] ,
         [ foaf:mbox <mailto:stefan.zander@univie.ac.at> ] ;
      ncal:location [ a geonames:Feature ; rdfs:label "Munich" ] .
   ] .
<http://sws.geonames.org/2761369/>
   a geonames:Feature ;
   rdfs:label "Vienna"@en .
```

FIGURE 5.7: Aggregated context models that constitute a context configuration instance

```
DESCRIBE ?c WHERE {
   { ?c rdfs:label ?l .
      ?l bif:contains "Vienna" .
      ?c rdf:type dbpedia-owl:Place .
   } UNION
   { ?c rdfs:label ?l .
      ?l bif:contains "Salzburg" .
      ?c rdf:type dbpedia-owl:Place .
   } UNION
   { ?c rdfs:label ?l .
      ?l bif:contains "Munich" .
      ?c rdf:type dbpedia-owl:Place .
   }
}
```

FIGURE 5.8: An example SPARQL query produced by the `DBpediaLocationDataProvider`

distributed across the Web on the basis of the e-mail addresses extracted from the context models of the `GoogleCalendarContextProvider`. This includes names, contact and location information, as well as personal interests of the user's prospective business partners. Also, it includes the social network of the meeting participants and is therefore valuable information for business negotiations as well as smalltalk.

A second data provider retrieves triples about people that are based near the user's current location by looking up resources that are `foaf:based_near` the current and future locations. In a business context, this information allows a knowledge worker or business consultant to increase the effectiveness of their trip by scheduling additional meetings with these persons without additional travel costs.

The `DBpediaLocationDataProvider` as third data provider returns additional data from the DBpedia data set about the user's current and future locations, by reusing the GeoNames URI

provided by the `GeonamesContextProvider`. By doing so, the user is automatically equipped with information about the locations they will visit, and about points of interests in their vicinity. Those data sets can be further interlinked with data sets from other services such as an event database or a review repository using the GeoNames URI. A code excerpt from the `DBpediaLocationDataProvider` is depicted in Figure 5.3.

From an initial analysis, we can expect a significant effect on the amount of potentially interested data that is to be replicated to a mobile device. For instance, the public DBpedia data set contains information about around 462,000 places. While no detailed information is available, from the overall size of the data set we can estimate that these places are described by around 88 million triples[15]. By analyzing the user's calendar and querying DBpedia for corresponding resources, this amount of data can be significantly reduced. For instance, if the context framework detects three locations in the user's calendar, it can convert them into a SPARQL query (cf. Figure 5.3) and query the DBpedia data set. In case the user's upcoming events within the next 72 hours take place in Vienna, Salzburg, and Munich, the corresponding query (cf. Figure 5.8) yields around 8,500 triples, which can be handled by common state-of-the-art smartphones (cf. Chapter 6).

All replicated data are persisted by the RDF content provider (cf. Section 4.6.2.12) that wraps and encapsulates well-defined interfaces for accessing or modifying locally deployed triple stores, data bases, or other storage systems (cf. Section 4.6.2). The RDF content provider per default either serializes RDF graphs into flat files, which is a very performant way for persisting RDF data directly on the local file system, however, accompanied with the loss of query functionality as those file-based serialization graphs need to be loaded to the memory first before a graph can be queried. Alternatively and as envisaged by the conceptual architecture (cf. Section 4.6), replicated RDF data are stored as RDF graphs into a custom triple store or into the default triple store that is backed by a SQLite database natively deployed in the Android operating system. The table layout of the default triple store applies the *normalized triple store* approach; i.e., it stores triples within a `Triple` table that holds references to separate tables for resources and literals. Moreover, it provides lightweight support for *named graphs*; therefore the relational schema contains a separate `Graph` table and allows to access data replicas on different levels of granularity.

Any application built on top of this framework is now enabled to directly access these data via the RDF content provider. It could, for instance, iterate over all resources that are typed as `foaf:Person` and provide a list of names and phone numbers, disburdening the user from the need to manually search for these data in case they will miss an appointment and needs to notify the participants. The proposed context-sensitive RDF data replication framework entirely hides all context processing steps: an application is presented with a simple view on the triple store which is at any time populated with context-relevant information replicated from related data sources.

5.4 Summary

In this chapter, we have presented aspects related to the development platform of our proof-of-concept prototype together with the class and interface structure of context and data providers. Our design approach relieves 3rd-party developers that aim to utilize the framework from the necessity of having to deal with processing and management-specific aspects; instead developers

[15] DBpedia 3.6 release notes: `http://blog.dbpedia.org/2011/01/17/dbpedia-36-released/`

can focus on key aspects related to the acquisition of contextual information or on the replication logic, i.e., the business logic for retrieving data from external data sources as a result of the analysis of the context configuration. However, we only presented a comparatively small excerpt of our proof-of-concept implementation but included references to more detailed and comprehensive elaborations of the framework's software architecture and the software-specific implementation of the framework's algorithms and concepts.

In the second part of this chapter, we have presented a case study that is based upon the information needs of a typical mobile knowledge worker or business consultant that is reliant on her mobile device for managing her business-related information assets. We implemented and introduced a number of context and data providers that acquire context-relevant information from the location sensor deployed on a mobile device as well as from the user's calendar and combines and aggregates such information in order to constantly provide the user with high-level information about her current location and the people she is likely to meet in the near future. As the framework is designed as an infrastructure that needs to be adapted to specific use cases and application scenarios, we focused on a use case that is centered around the provision of location-based information as this type of context information is most relevant for mobile users (cf. [BZD02, Bar03, RVW05, HSMY08, SVLO$^+$11]). The case study shows that a significant amount of information that exhibits no direct relationship to the user's current and future context does not need to be considered by the user when browsing for information that is of no relevance to her; this in consequence minimizes cognitive load and explicit user attention when searching for information.

Chapter 6

Evaluation of the Processing Efficiency of RDF Data Replicas

"The problem is never how to get new, innovative thoughts into your mind, but how to get old ones out. Every mind is a building filled with archaic furniture. Clean out a corner of your mind and creativity will instantly fill it."

Dee Ward Hock (* 1929)

In the resource-limited context of mobile devices, the efficient processing of RDF data in general and RDF data replicas in particular for the proactive provision of context-relevant information is crucial. In order to obtain insights related to the processing efficiency of local RDF data replicas on modern mobile platforms using the proposed context-sensitive replication framework as described in Chapter 4, we have carried out a performance evaluation based on the three existing mobile RDF frameworks *Androjena*, *μJena*, and *Mobile RDF* (cf. Section 3.3.2), conducted on four technically differing mobile devices (cf. Table 6.1). We analyze the runtime behavior of typical RDF processing operations applied to local RDF data replicas to assess whether data replicas can be processed on current mobile devices in reasonable time using currently available RDF frameworks. The analysis comprises benchmarks related to parsing, storing, and querying RDF documents, i.e., in-memory representations of RDF data replicas, as well as adding and removing distinct elements (subjects, predicates, objects) to/from RDF graphs of varying sizes. Additionally, we also conducted benchmarks related to the creation of in-memory RDF models, as these are necessary for the creation of the single context models as well as for the transformation and aggregation of those models into a new context configuration instance (cf. Section 4.3.8) to which reasoning and consolidation heuristics are applied in subsequent processing steps.

The evaluation exclusively concentrates on the three currently available Java-based RDF frameworks for mobile platforms introduced and analyzed in Section 3.3.2 that are compatible to the Android SDK[1]. We excluded the XML/RDF parsers introduced in Section 3.3.1 as they lack fundamental RDF processing capabilities as discussed in Section 3.3.4. Furthermore, we did not include the two RDF infrastructure frameworks[2] *RDF on the Go* and *SWIP* that have been introduced in Section 3.3.3 in this evaluation since they either exist as an implementation of a

[1] Android SDK: `http://developer.android.com/sdk/index.html`

[2] RDF infrastructure frameworks offer additional functionality such as querying or persistence on top of existing RDF frameworks.

specific platform-dependent technology (SWIP) or have been released after our evaluation has been conducted (RDF on the Go).

We specifically focused on the Android platform since it is an open platform and, compared to other mobile platforms, incorporates a number of advantages and unique concepts that have been utilized for the acquisition and processing of context-relevant information (see Section 5.1 for a discussion). Therefore, we exclusively concentrated on those RDF frameworks that are available for or are compatible to the Android platform as this is the development platform of our prototype. Additionally, the growing popularity of Android as one of the leading mobile development platforms to date is unmistakably evident.

All evaluation benchmarks have been conducted on four technically varying mobile devices. Those devices have been carefully chosen for the evaluation as they represent different device classes that correspond to distinct mobile market segments (entry-level, middle-class, and high-level) and differ in their technical capabilities, hardware and software features, and runtime environments. This selection allows for extrapolating on how well the replication framework performs on devices pertaining to a particular market segment in combination with a particular RDF framework. Additionally, it allows us to generally ascertain the processing efficiency of small and comparably large data replicas containing several thousand RDF triples on technically varying devices from different market segments using a specific RDF framework. To verify and substantiate our findings, we included two technically equivalent devices in this evaluation, the Samsung Galaxy S I9000 and the Dell Streak 5[3], to gather insights whether the replication framework exhibits a similar performance behavior on devices equipped with different CPUs.

The remainder of this Chapter is structured as follows: in Section 6.1, we give an overview of the test environment and introduce the devices used throughout this evaluation. Details regarding our test respectively benchmark setup are outlined in Section 6.2, which also provides some details of the test data used in the benchmarks to simulate live data being replicated in real-world conditions. The results of each benchmark are presented and thoroughly discussed in Section 6.3, which is structured according to the operations being analyzed in the course of this evaluation. This section also contains detailed graphical illustrations of the results acquired in each benchmark. Finally, we discuss and summarize our findings in Section 6.4.

6.1 Test Environment

In the following, we provide a short overview of the devices included in the evaluation. Further technical details of each device can be found in the corresponding footnotes.

The HTC G1[4], released in 2008, was one of the first commercially available Android devices and represents the entry-level device class. It contains a 32-bit Qualcomm MSM7201A RISC CPU that runs with a clock speed of 350 MHz. Tests on this device were performed with the standard memory capacity of 192 MB under the Android operating system version 1.6 update 4.

The Motorola Milestone[5] was released in December 2009 and represents the middle-class of Android devices. It runs on a 32-bit TI OMAP3430 Superscalar ARM Cortex-A8 RISC CPU with a nominal clock speed of 600 MHz. The evaluation benchmarks were performed with the standard memory capacity of 256 MB under the Android operating system version 2.1 update 1.

[3]We use the denomination 'Dell Streak' in the course of this chapter.
[4]Technical specification of the Android HTC G1: `http://www.htc.com/www/product/g1/specification.html`
[5]Technical specification of the Motorola Milestone: `http://www.motorola.com/Consumers/XW-EN/Consumer-Products-and-Services/Mobile-Phones/ci.Motorola-MILESTONE-XW-EN.alt`

Table 6.1: Overview of the Android Devices' Specification

	HTC G1 (1)	**Motorola Milestone** (2)
Processor	Qualcomm MSM7201A™	TI OMAP3430 ARM Cortex A8
Clock speed in MHz	350 MHz	600 MHz
Memory Capacity (RAM)	192 MB	256 MB
OS Version	Android 1.6-update4	Android 2.1-update1
Release	09/2008	12/2009
Market segment	Entry-level	Middle-level
	Samsung Galaxy S I9000 (3)	**Dell Streak 5** (4)
Processor	Qualcomm S5PC111	Qualcomm QSD 8250 Snapdragon
Clock speed in GHz	1 GHz	1 GHz
Memory Capacity (RAM)	512 MB	512 MB
OS Version	Android 2.2	Android 2.2.1
Release	06/2010	10/2010
Market segment	Upper-/Top-level	Upper-/Top-level

The Samsung Galaxy S I9000[6] smartphone, which was released in Summer 2010, represents the former latest generation of mobile devices and thus the high- or upper-level market segment. It uses a Qualcomm S5PC111 ARMv7-compatible CPU named "Hummingbird" with a nominal clock speed of max. 1 GHz paired with a PowerVR SGX540 GPU chip. This device uses 512 MB main memory and runs the Android system version 2.2.

The Dell Streak 5[7] also adheres to the high- or upper-level mobile market segment and represents a hybrid combination of smartphone and tablet device with a 5-inch display. It was released in October 2010 and incorporates a 1GHz Qualcomm QSD 8250 Snapdragon processing unit that features an ARMv7 instruction set. The CPU is manufactured in a 65 nanometer semiconductor technology process and equipped with an Adreon 200 GPU chip. The Dell Streak has 512 MB ROM together with 512 MB main memory built in. All tests on this device were performed under the Android operating system version 2.2.1.

6.2 Test Setup

An important factor for the efficient processing of RDF data replicas is the time needed to create, load, parse, and store an RDF model in-memory, as this is usually the basis for further computation, analysis, inference, or transmission of data over a network. We therefore analyzed the creation, parsing, storage, and query time for RDF models[8] of various sizes and also ascertained the execution time of modification operations such as the insertion and removal of distinct elements. In particular, we measured such operations for RDF models of the following sizes:

```
10, 20, 50, 100, 200, 500, 1000, 2,000, 5,000, 10,000, 20,000, 50,000, 100,000
```

This list represents the different model sizes that are involved in the context processing and data replication tasks performed by the presented framework described in Chapter 4.6. Typically, a

[6] Technical specification of the Samsung Galaxy S I9000: `http://pdadb.net/index.php?m=specs&id=2298&c=samsung_gt-i9000_galaxy_s_16gb`

[7] Technical specification of the Dell Streak 5: `http://www.dell.com/de/p/mobile-streak/pd`

[8] We use the terms '*RDF model*' or '*RDF graph*' to refer to in-memory representations of RDF data replicas.

single context provider emits very small models in the range of 10 to 100 triples, while a complete context configuration model that has been aggregated from the context descriptions of the single context providers may have several hundred to thousand triples in total. Data that are replicated from external sources may in principle be of arbitrary size, therefore we have scaled our tests up to 100,000 triples in a single RDF model.

The distribution of distinct subject, predicate, and object nodes has been estimated based on an analysis of the 2009 Billion Triple Challenge data set [Sch10]. In these data we can observe that typically RDF data sets have a very high number of distinct object values and a low number of distinct predicates, while the number of distinct subjects ranges in between these boundaries. All benchmarks were performed on the mobile devices during regular usage of a device where the usual system processes were running in parallel to our tests.

6.2.1 Test data

To obtain results that reflect real-world replication scenarios, we make use of data sets hosted within the DBpedia project for this evaluation. The DBpedia database contains lots of useful information that might be replicated to a mobile device based on the user's current context and information needs. These data sets have been extracted and aggregated from the Wikipedia project and are available as structured information being published under the GNU General Public license[9] wherefore they are used for the simulation of data that are replicated under real-world conditions. Figure 6.1 depicts an excerpt of the DBpedia test data set used for this evaluation being serialized in N-Triples format, which represents relevant information about well-known persons extracted from a specific database dump[10] of the DBpedia database.

This data set contains information about persons being identified using a DBpedia-specific URI (e.g. `http://dbpedia.org/resource/Aristotle`) and relevant information about them encoded in predicates and objects represented through terms from the DBpedia ontology[11] and other ontological vocabularies such as Dublin Core[12] (e.g. `http://purl.org/dc/elements/1.1/description`), or as plain or typed literal values (e.g. `"Greek philosopher"@en`). This information has been extracted from different Wikipedia pages and interlinked with related data from other data sets.

6.2.2 Preparation of Test Data

The data sets retrieved from the DBpedia dump have been split in several data chunks of varying sizes ranging from 10 to 100,000 triples and serve as representations of different data replicas being processed during the execution of the benchmark instances, i.e., the data contained in a data chunk can be considered as data replicated to the mobile device by one or more data providers. Before data chunks are transferred to a mobile device, they were transformed into different RDF serialization formats using the tool *Any23*[13] in order to execute the benchmarks with all serialization formats supported by an RDF framework; for instance, the Androjena framework supports the three major RDF formats *RDF/XML*, *N3*, and *N-Triples* (cf. Figure 3.1). Having the test data available in multiple serialization formats allows us to obtain information whether differences in terms of parsing and saving performance depending on a concrete serialization format exist for a framework and which format yields best performance results.

[9] Information about the GNU General Public license: `http://www.gnu.org/copyleft/gpl.html`
[10] The database dump was retrieved from: `http://downloads.dbpedia.org/3.6/en/persondata_en.nt.bz2`
[11] The DBpedia Ontology: `http://wiki.dbpedia.org/Ontology`
[12] The Dublin Core® Metadata Initiative: `http://dublincore.org/documents/dcmi-terms/`
[13] Any23 tool: `http://developers.any23.org/`

```
[...]
<http://dbpedia.org/resource/Aristotle>
   <http://dbpedia.org/ontology/deathPlace>
   <http://dbpedia.org/resource/Chalcis> .
<http://dbpedia.org/resource/Aristotle>
   <http://dbpedia.org/ontology/birthPlace>
   <http://dbpedia.org/resource/Stageira> .
<http://dbpedia.org/resource/Aristotle>
   <http://purl.org/dc/elements/1.1/description>
   "Greek philosopher"@en .
<http://dbpedia.org/resource/Aristotle>
   <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
   <http://xmlns.com/foaf/0.1/Person> .
<http://dbpedia.org/resource/Aristotle>
   <http://xmlns.com/foaf/0.1/name> "Aristotle"@en .
<http://dbpedia.org/resource/Abraham_Lincoln>
   <http://dbpedia.org/ontology/deathPlace>
   <http://dbpedia.org/resource/Washington,_D.C.> .
<http://dbpedia.org/resource/Abraham_Lincoln>
   <http://dbpedia.org/ontology/deathDate>
   "1865-04-15"^^<http://www.w3.org/2001/XMLSchema#date> .
<http://dbpedia.org/resource/Abraham_Lincoln>
   <http://dbpedia.org/ontology/birthPlace>
   <http://dbpedia.org/resource/Kentucky> .
<http://dbpedia.org/resource/Abraham_Lincoln>
   <http://dbpedia.org/ontology/birthDate>
   "1809-02-12"^^<http://www.w3.org/2001/XMLSchema#date> .
<http://dbpedia.org/resource/Abraham_Lincoln>
   <http://purl.org/dc/elements/1.1/description>
   "16th President of the United States of America"@en .
<http://dbpedia.org/resource/Abraham_Lincoln>
   <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
   <http://xmlns.com/foaf/0.1/Person> .
[...]
```

FIGURE 6.1: An excerpt of the DBpedia test data set used for the performance evaluation

In order to eliminate technical discrepancies resulting from differing network connection quality and network latencies as well as different storage infrastructures and other aspects, and to make the test data constantly available during the benchmark processes, they have been copied to an SD card installed in each device prior to each benchmark. Before a concrete instance of a benchmark is executed, the data replicas are copied from the SD card to the internal non-volatile memory of a device from where they are then parsed and transformed into workable in-memory models. This procedure allows us to neutralize technological differences that exist between SD cards in terms of access times as well as read-/write performance.

6.2.3 Recording of Benchmark Results

All benchmark results as well as failures that emerged during the execution of a benchmark instance were logged in separate log files in order to automatically reprocess and analyze them afterwards using statistical applications such as *MATLAB*[14], *R*[15], or *DataGraph*[16]. This allows for calculating several statistic measures such as the arithmetic mean, the standard deviation,

[14]MATLAB: `http://www.mathworks.com/products/matlab/`
[15]The R project for statistical computing: `http://www.r-project.org/`
[16]DataGraph: `http://www.visualdatatools.com/DataGraph/`

as well as the processed RDF triples per one second, which is a normalized representation of the general RDF processing performance of a framework.

All benchmarks were performed by a special benchmark application that executes the corresponding framework functions and RDF processing operations introduced in Section 6.3. The benchmark application was locally deployed on each device and runs in parallel to the replication framework as well as system and user processes. It initiates and monitors the execution of replication tasks as well as modification operations on RDF data replicas and stores the results in corresponding log files. Those files are hosted solely on the SD card to avert influences on consecutive benchmarks as well as benchmarks that are executed on the internal memeory. The general workflow of the benchmark process is conceptually outlined in Algorithm 10.

Algorithm 10: Conceptual workflow of the evaluation benchmarks

Data: benchmark method m, data replica r, serialization format f, RDF framework fw
Result: benchmark results of each run stored in corresponding log file

```
Set Frameworks ← ⟨fw_1, ..., fw_k⟩ ;
foreach fw ∈ Frameworks do
    Set Formats ← IdentifySupportedSerializationFormats(⟨f_1, ..., f_l⟩) ;
    foreach f_i ∈ Formats do
        Set Replicas ← CopyDataChunksToInternalMemory(⟨r_1^{f_i}, ..., r_m^{f_i}⟩) ;
        InitializeBenchmarkInstance(fw,f_i,m,...) ;
        Set logfile ← CreateLogfile(fw,f_i) ;
        foreach r_j^{f_i} ∈ Replicas do
            LoadReplicaIntoMainMemory(r_j^{f_i}) ;
            while current run < number of iterations + 1 do
                InitializeCurrentInstance(...) ;
                Set result ← ExecuteBenchmark(m, r_j^{f_i}) ;
                StoreResultInLogfile(i, result, logfile) ;
                CleanUp → InitiateGarbageCollector() ;
            end
            RemoveReplicaFromMainMemory(r_j^{f_i}) ;
        end
        CloseLogfile(logfile) ;
        RemoveDataChunkFromInternalMemory(⟨r_1^{f_i}, ..., r_m^{f_i}⟩) ;
    end
end
```

For each framework, device, serialization format, and operation (cf. Section 6.3), we measured the total amount of time needed in milliseconds to finish an operation successfully. From these measurements we calculated the standard deviation between different benchmark iterations for each size as well as the number of triples that the particular combination of a device and a framework is able to process within one second. Each benchmark was repeated ten times to minimize variations among consecutively conducted benchmark instances.

All benchmarks had been executed in normal operation mode after restarting the devices to ensure identical runtime conditions. This setting allows us to simulate real-world conditions. At the end of each benchmark, all files and data that had been created during a test run were deleted and the test environment was reseted to avert influences on consecutive benchmarks.

6.3 Results

In order to evaluate the practical applicability of our approach, we conducted the subsequently listed benchmarks. Each benchmark is described in detail in a separate section (given in brackets) followed by a discussion and interpretation of the acquired results. In addition to usual operations related to the parsing and serialization of data being replicated to the mobile device, we also analyzed operations that concern modifications applied to RDF data replicas since data that are replicated according to specific contextual constellations are usually modified in subsequent steps rather than merely stored locally. In particular, we analyzed the runtime behavior of the following operations, which are extensively discussed in the listed sections:

- Parsing RDF data replicas (Section 6.3.1)
- Storing RDF data replicas (Section 6.3.2)
- Inserting data in RDF data replicas (Section 6.3.3)
- Removing data from RDF data replicas (Section 6.3.4)
- Retrieving distinct elements from RDF data replicas (Section 6.3.5)
- Constructing in-memory RDF graphs (Section 6.3.6)

At the beginning of each section, we provide a short overview of the benchmark's setting, its objectives, as well as the specific aspects that had been analyzed. This overview is complemented by an algorithmic description of the major steps involved in conducting a specific operation being benchmarked. For each benchmark, we discuss, interpret, and summarize the results w.r.t. the involved devices and frameworks, and visualize the acquired values in form of plotted data graphs[17] located in the corresponding sub sections. The detailed values of each benchmark instance together with further statistical calculations such as standard deviation and processed triples per second ratios can be found in the respective tables located in Appendix A.

6.3.1 Parsing RDF Data Replicas

In the parsing benchmark, we measured the total amount of time needed to parse and create a workable in-memory representation of an RDF data replica. These measurements allow us to obtain insights regarding the parsing performance of each framework on different classes of mobile devices. Furthermore, it allows us to assess whether large data replicas can be transformed and processed in reasonable amounts of time. If a framework supports multiple serialization formats, we repeated the parsing benchmark for each format separately to see whether there exist significant differences in parsing performance across serialization formats.

During an instance of a benchmark run, a data replica of specific size and serialization format is loaded from the internal memory to the RAM and parsed into a framework-specific in-memory model to which query, insertion, or removal operations can be applied (the runtime behavior of such operations is discussed in the Sections 6.3.3, 6.3.4, and 6.3.5). The workflow of the parsing benchmark is conceptually described in Algorithm 11.

[17] However, in some data graphs the labels `RDF/XML`, `N3`, and `N-TRIPLE` refer to the different serialization formats supported by the Androjena framework. For readability issues we excluded the name 'Androjena' and just referred to the respective serialization formats.

Algorithm 11: Parsing data replicas

```
Set Formats ← IdentifySupportedSerializationFormats(⟨f_1, ..., f_l⟩) ;
foreach f_i ∈ Formats do
    Set Replicas ← CopyDataChunksToInternalMemory(⟨r_1^{f_i}, ..., r_m^{f_i}⟩) ;
    foreach r_j^{f_i} ∈ Replicas do
        CopyReplicaToInternalMemory(r_j^{f_i}) ;
        while current run k < number of iterations + 1 do
            Set t_0 ← System.getCurrentTimeMillis() ;
            Set model ← ParseDataReplica(LoadDataReplica(r_j^{f_i})) ;
            Set t_1 ← System.getCurrentTimeMillis() ;
            WriteResultsToLogfile(t_1 − t_0, f_i, r_j^{f_i}, k, !model.isEmpty()) ;
            CleanUp → InitiateGarbageCollector() ;
        end
        RemoveReplicaFromInternalMemory(r_j^{f_i}) ;
    end
end
```

The results of the parsing benchmark compartmentalized by framework and serialization format are depicted in Figure 6.2, 6.3, and 6.4. The plotted graphs represent the parsing performance measured in processed triples per second ratios for transforming RDF data replicas of particular sizes into workable in-memory representations of RDF graphs. The size of the data replicas is marked on the x-axis; the processed triples per second ratios are displayed along the y-axis. In addition, we also included an illustration of the parsing performance compartmentalized by device at the end of this section, which is depicted in Figure 6.5.

6.3.1.1 Androjena

Using the Androjena framework, we were able to parse data replicas containing a maximum of 5,000 triples[18] on the HTC G1, 20,000 triples on the Motorola Milestone, 50,000 triples on the Samsung Galaxy S I9000, and 100,000 triples on the Dell Streak, depending on the serialization format. The Androjena framework, as opposed to μJena and Mobile RDF, supports the three most popular RDF serialization formats RDF/XML, N-Triples, and N3 (see Section 3.3.4) wherefore benchmarks have been executed for each format separately.

Androjena yields best parsing performance when RDF replicas are serialized using the N-Triples format, followed by N3 and RDF/XML (see Figure 6.2). The cumulated average storage performance per serialization format across all replicas and devices amounts to 695.85 triples per second using the N-Triples format, 390.02 triples per second using the N3 format, and 292.41 triples per second using the RDF/XML format. Analyzing the cumulated average processed triples per second ratios over all devices revealed that N3 serialized data replicas are parsed by the factor 1.33 faster than replicas being serialized in the RDF/XML format; by parsing data that is serialized in the N-Triples format instead of N3, the overall parsing performance over all devices growth by the factor 1.78.

Looking specifically at the parsing performance per serialization format and device, the average processed triples per second performance across all data replicas serialized in the RDF/XML-format account to 88.22 triples per second on the HTC G1 (see Table A.1), 279.74 triples per

[18] During some benchmarks, we were able to process RDF data replicas containing up to 10,000 triples (e.g., in Section 6.3.5.1).

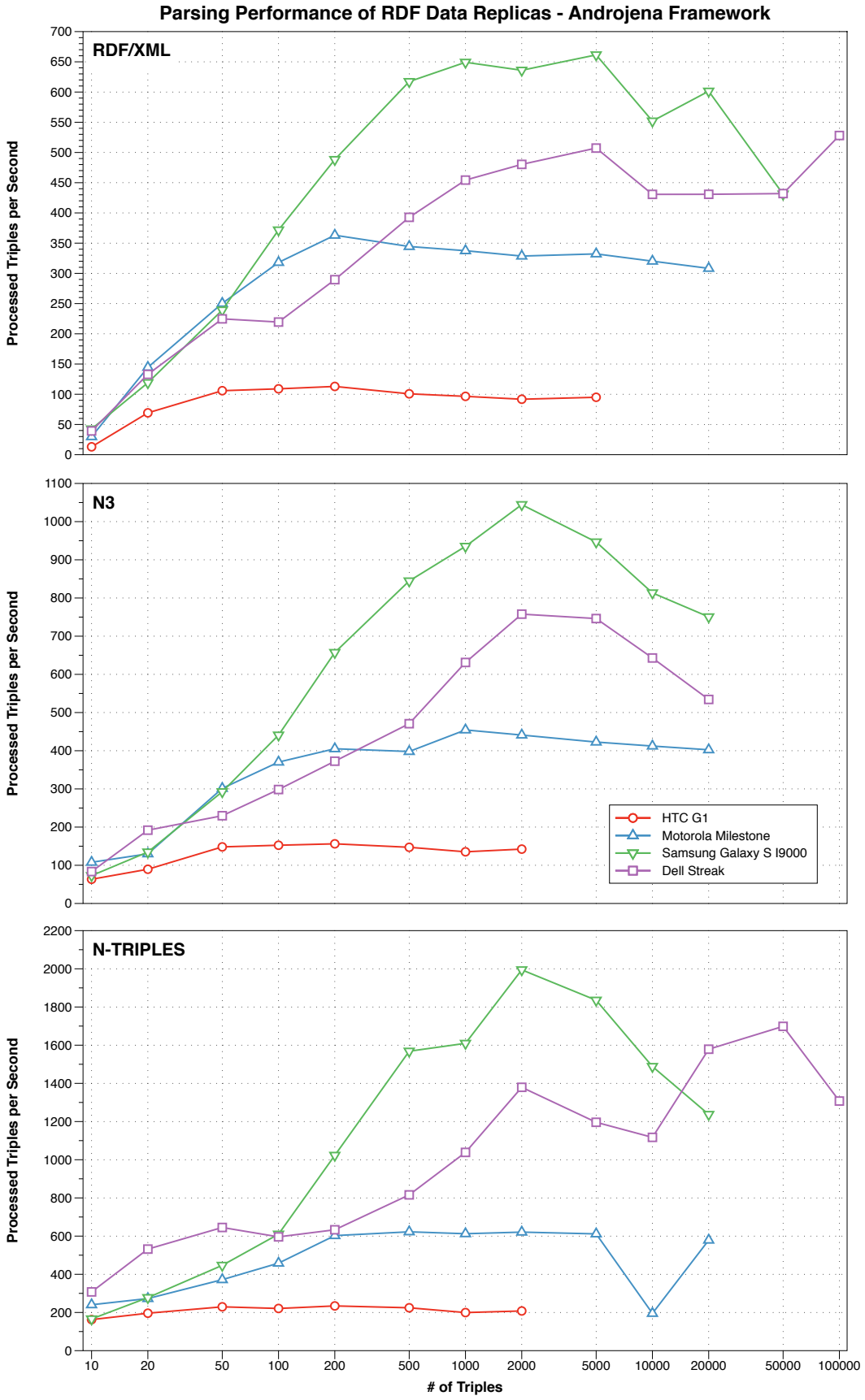


FIGURE 6.2: Parsing performance of the Androjena framework depending on the serialization format

second on the Motorola Milestone (see Table A.2), 450.77 triples per second on the Samsung Galaxy S I9000 (see Table A.3), and 350.93 triples per second on the Dell Streak (see Table A.4). Using the N3 serialization format, average parsing performance across all data replicas grows by the factor 1.46 on the HTC G1, by the factor 1.25 on the Motorola Milestone, by the factor 1.39 on the Samsung Galaxy S I9000, and by the factor 1.28 on the Dell Streak. Parsing data replicas serialized in N-Triples-format, triples per second ratios grows by the factor 1.62 on the HTC G1, factor 1.35 on the Motorola Milestone, factor 1.76 on the Samsung Galaxy S I9000, and factor 2.19 on the Dell Streak. As these numbers indicate, the slowest parsing performance per serialization format was measurable for RDF/XML serialized data replicas (see Figure 6.2). Additionally, the lowest parsing performance per device was measured on the HTC G1 across all serialization formats (see Figure 6.5).

The parsing performance of Androjena is significantly slower for smaller data replicas irrespectively of the serialization format and device; for instance, the average triples per second ratios across all serialization formats almost grow by the factor 13.05 on the Samsung Galaxy S I9000 when replicas containing 2,000 triples instead of 10 triples are parsed. However, the increase in processed triples per second performance is observable on all devices and formats, although less distinctive on devices from the low- and middle-market segment. As a consequence, for comparably small data replicas the sizes of which range between 10 and 200 triples (usually on faster devices, these numbers scale up to 2,000 triples), we could notice a constant increase in parsing performance on all devices and for all serialization formats. The highest parsing performance could be measured for data replicas whose sizes lie between 1,000 and 5,000 triples depending on the device and serialization format.

The parsing performance in terms of triples per second ratios begins to decrease again for larger data replicas with more than 2,000 to 5,000 triples, depending on the device and serialization format. However, we observed a greater variance in processed triples per second ratios on the two more powerful devices, which was not that distinctive on the HTC G1 and the Motorola Milestone where processed triples per second ratios scale more moderately and remain relatively constant for larger data replicas. This behavior might be attributed to the weaker CPU performance and the smaller physical main memory installed in the HTC G1 and the Motorola Milestone. Additionally, we could see that the parsing performance of the Androjena framework scales relatively well with available processor power as well as with available physical main memory although it was not possible to parse data replicas with 100,000 triples on the Samsung Galaxy.

In summary, Androjena attains the highest parsing performance on the Samsung Galaxy S I9000; the average parsing performance across all data replicas and serialization formats was approx. 1.23 times faster than on the Dell Streak. Comparing the average parsing performance of Androjena across all serialization formats on the Samsung Galaxy S I9000 to the HTC G1 and the Motorola Milestone, parsing operations finish by the factor 5.14 faster on the Samsung Galaxy compared to the HTC G1 and by the factor 1.99 faster compared to the Motorola Milestone. In consequence, the average processed triples per second ratios across all data replicas and serialization formats almost double between devices from different mobile market segments using the Androjena framework.

In summary, Androjena benefits from increased processing power and yields best parsing results in terms of processed triples per second ratios with RDF graphs containing around 2,000 triples. However, we were not able to notice a remarkable difference between the different serialization formats on newer mobile device with graphs smaller than 100 triples, i.e., perceivable differences in benchmark results among different serialization formats can first be noticed on newer mobile devices for graphs with more than 100 triples.

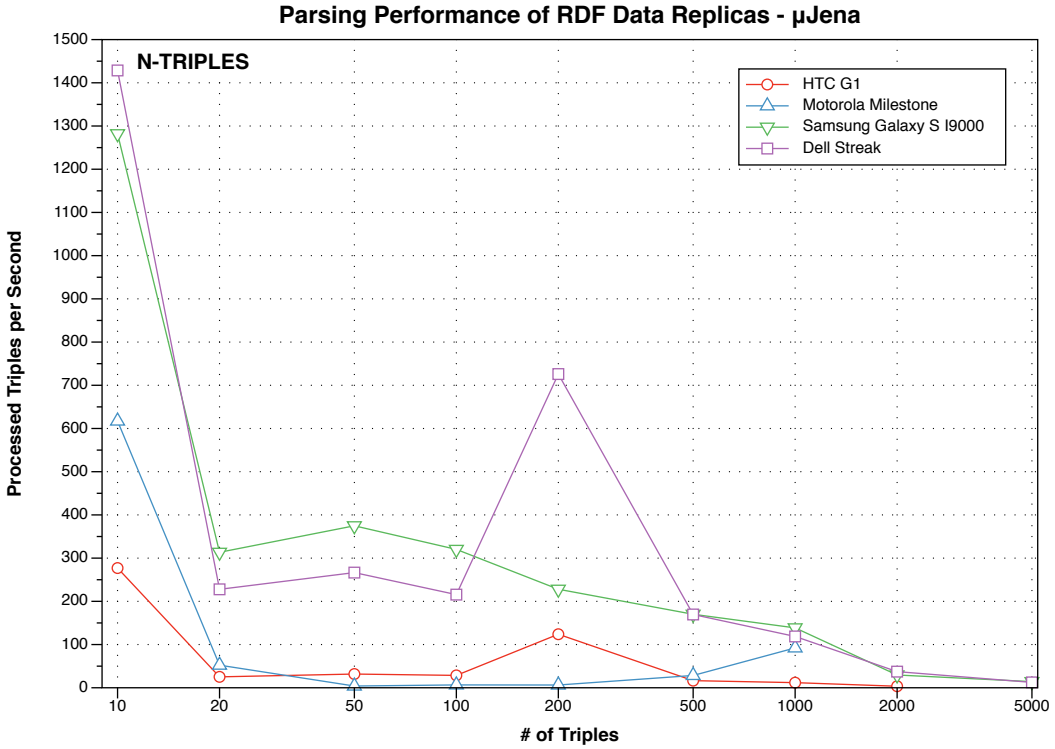


FIGURE 6.3: Parsing performance per device using the μJena framework

6.3.1.2 μJena

The size of data replicas that can be parsed with the μJena framework is significantly lower compared to the other RDF frameworks: in total, we were not able to parse data replicas with more than 5,000 triples on any device. On less powerful devices, however, data replicas have to be reduced to 1,000 triples (Motorola Milestone) respectively 2,000 triples (HTC G1). Furthermore, μJena is capable of processing N-Triples-serialized data only.

A dramatic decrease in parsing performance on all devices was observable for parsing data replicas that contain 20 or more triples (see Figure 6.3); for instance, processed triples per second performance diminish from 617.28 triples per second for parsing 10 triples to 52.27 triples per second for parsing replicas containing 20 triples on the Motorola Milestone. This dramatic decrease was also visible on the two upper-segment devices: for identical data replica sizes, parsing performance dropped by the factor 6.27 on the Dell Streak and by the factor 4.09 on the Samsung Galaxy S I9000. When data replicas containing more than 20 triples are processed, the decrease was still measurable but not that dramatic. However, decreases in parsing performance were less distinctive on more powerful devices but still noticeable. In this respect, we can observe that there exists a strong dependency in parsing performance and data replica sizes using the μJena framework as illustrated in Figure 6.3. Moreover, an extraordinary behavior was observable on the Motorola Milestone where the parsing performance starts to increase for data replicas containing more than 200 triples. This behavior was not visible on the other devices.

In general, the average parsing performance in terms of processed triples per second ratios across all data replicas amounts to 64,78 triples per second on the HTC G1 (see Table A.1), 115.30 triples per second on the Motorola Milestone (see Table A.2), 318.99 triples per second on the Samsung Galaxy S I9000 (see Table A.3), and 355.83 triples per second on the Dell Streak

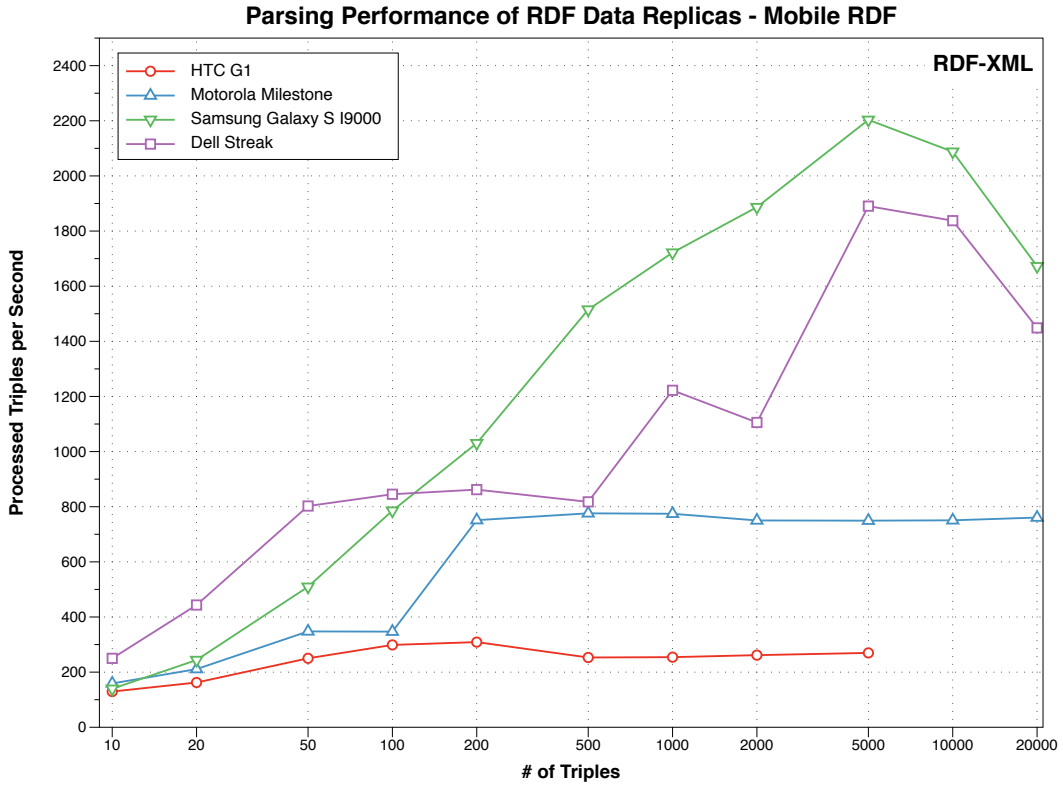


FIGURE 6.4: Parsing performance per device using the Mobile RDF framework

(see Table A.4). However, the fastest parsing performance on average over all data replicas was measurable on the Dell Streak, which lies by the factor 1.12 above the numbers acquired on the Samsung Galaxy S I9000. Compared to the Motorola Milestone and the HTC G1, the parsing performance on the Dell Streak increases by the factor 3.09 and 5.49 respectively. In addition, parsing performance on the Motorola Milestone increases by the factor 1.78 compared to the HTC G1. As these numbers indicate, the parsing performance of the μJena framework correlates with available processor power although it does not make efficient use of the larger physical main memory installed on the two upper-level devices. However, significant differences according to the variance of the acquired processed triples per second ratios among all devices were not identifiable.

In summary, μJena scales reasonably well with available processing power and yields best parsing results in terms of triples per second ratios with very small RDF graphs containing not more than 10 triples. With growing data replicas, parsing performance drops considerably; for instance, the triples per second ratios differ by the factor 115.10 between the smallest data replica containing 10 triples and the largest data replica containing 5,000 triples on the Dell Streak. Altogether, this behavior renders μJena inappropriate for processing larger data replicas.

6.3.1.3 Mobile RDF

The size of data replicas that can be parsed with the Mobile RDF framework accounts to 5,000 triples on the HTC G1 and 20,000 triples on the other devices. Moreover, Mobile RDF is capable of processing RDF/XML-serialized RDF documents only.

The average parsing performance across all data replicas serialized in the RDF/XML-format account to 243.15 triples per second on the HTC G1 (see Table A.1), 579.77 triples per second on the Motorola Milestone (see Table A.2), 1,254.22 triples per second on the Samsung Galaxy S I9000 (see Table A.3), and 1,047.80 triples per second on the Dell Streak (see Table A.4).

These numbers indicate that the Mobile RDF framework performs parsing operations most rapidly on the Samsung Galaxy S I9000 (see Figure 6.4): the average parsing performance across all data replicas was approximately 1.20 times higher than on the Dell Streak. Comparing the parsing performance of Mobile RDF on the Samsung Galaxy S I9000 to the HTC G1 and the Motorola Milestone, parsing operations finish by the factor 5.16 faster on the Samsung Galaxy compared to the HTC G1 and by the factor 2.16 faster compared to the Motorola Milestone. These results indicate that Mobile RDF scales reasonably well with available processing power: the average triples per second ratios across all data replicas almost double between devices from different mobile market segments using the Mobile RDF framework. Furthermore, we could identify a stronger growth in triples per second rations on the two more powerful devices, which was not that distinctive on the HTC G1 and the Motorola Milestone.

In total, Mobile RDF achieves best parsing results in terms of triples per second ratios on the Samsung Galaxy S I9000 with RDF graphs containing around 5,000 triples. In general, the highest triples per second ratios have been measured for parsing comparably large data replicas containing between 5,000 and 10,000 triples. With growing data replica sizes, parsing performance increase steadily and reaches its maximum when parsing 5,000 triples on the two upper-level devices; however, parsing performance starts to decrease for replicas containing 10,000 or 20,000 triples (see Figure 6.4).

The variance in processed triples per second ratios as visible on the μJena framework was not that distinctive on the Mobile RDF framework; for instance, the triples per second ratios differ by the factor 15.78 between to lowest and highest measured ratio on the Samsung Galaxy S I9000 and by the factor 7.56 on the Dell Streak. On the μJena framework, by way of comparison, these factors count up to 90.43 on the Samsung Galaxy and 115.10 on the Dell Streak. Comparing the variance between the lowest and highest processed triples per second values on the two less powerful devices reveals that parsing performance differs by the factor 2,38 on the HTC G1 and by the factor 4,88 on the Motorola Milestone. As a result, we can see that the variance in processed triples per second ratios is less distinctive on devices from the lower- and middle-market segment compared to devices placed at the upper-market segment.

6.3.1.4 Summary

Androjena, μJena, and Mobile RDF all scale reasonably well with available processing power where Mobile RDF turns out to be the fastest RDF framework in terms of parsing performance, especially for larger RDF graphs with more than 200 to 500 triples. This behavior was more distinctive on less powerful devices such as the HTC G1 or the Motorola Milestone but dissolved on recent, more powerful devices such as the Samsung Galaxy or the Dell Streak (see Figure 6.5). The best performance results could be measured with RDF graphs containing around 5,000 to 10,000 triples on the Samsung Galaxy S I9000. By contrast, μJena yields best performance results with very small data replicas and loses substantial processing performance with increasing data replica sizes. This considerable drop in parsing performance was observable on all devices (see Figure 6.5) and renders μJena inappropriate for the processing of data replicas. Considering the different serialization formats supported by Androjena, the best parsing results were measured with data replicas serialized in the N-Triples format followed by N3 and RDF/XML.

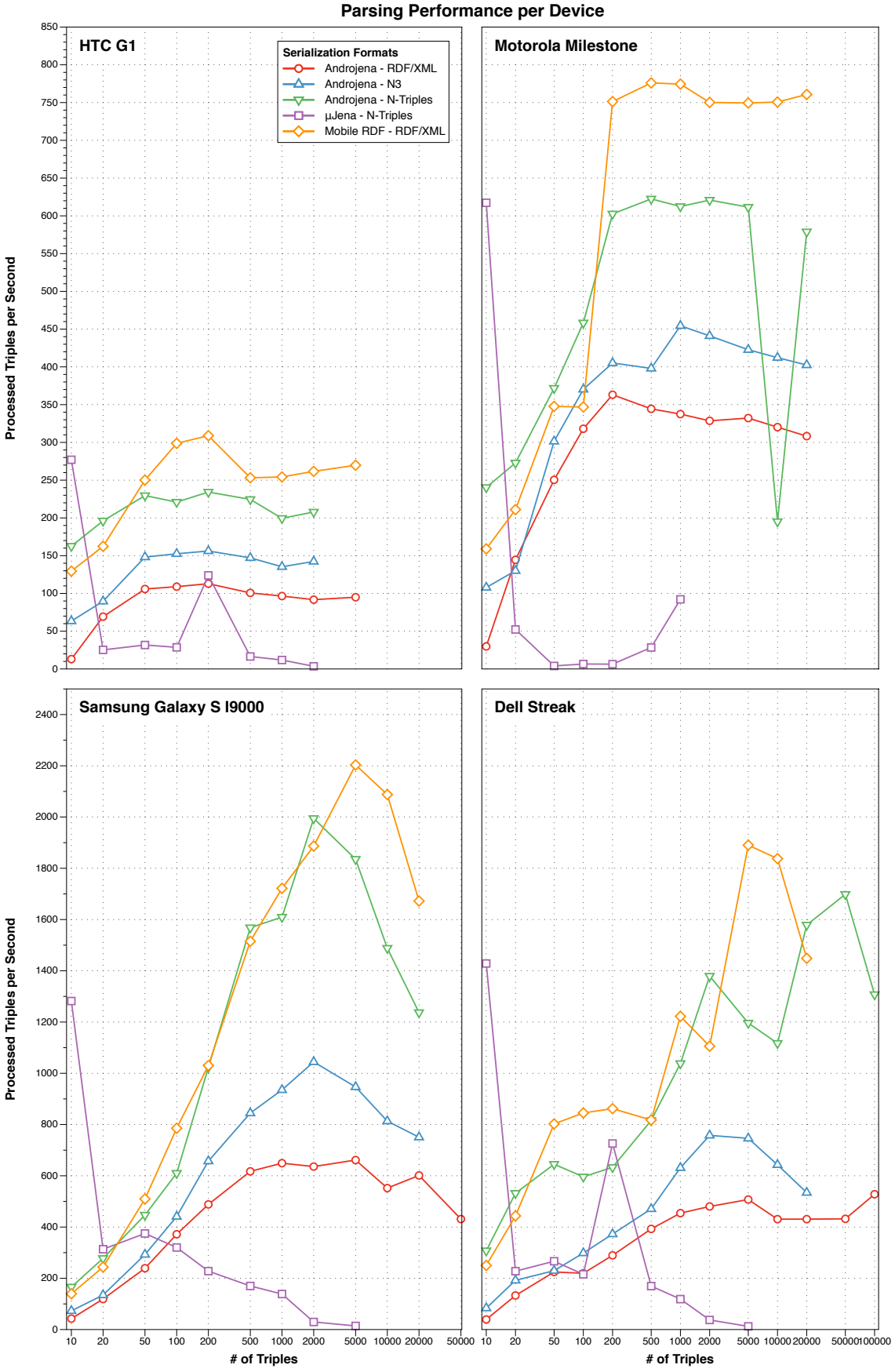


FIGURE 6.5: Parsing performance compartmentalized by device and serialization format

6.3.2 Serialization and Storage of RDF Data Replicas

In the serialization and storage benchmark, we measured the total amount of time needed to store data replicas on the local filesystem of a mobile device using a specific mobile RDF framework and serialization format. We recorded the time the serialization operation takes to transform a workable in-memory representation of an RDF data replica into a concrete serialization format and write it into a file object on the local filesystem. These measurements allow us to obtain insights regarding the storage performance of the available mobile RDF frameworks on different classes of mobile devices, particularly for large data replicas. Furthermore, we also recorded the file size of each stored replica depending on the supported serialization formats to see whether there exist significant differences in storage and serialization performance across different serialization formats and how the physical file size of a stored data replica correlates to the measured runtimes of a storage operation. If a framework supports multiple serialization formats, the storage benchmark has been conducted for each format separately.

For the storage and serialization benchmark, RDF data replicas had been copied from the SD card to the internal memory from which they were parsed and transformed into workable in-memory models before the serialization and storage operations of a framework are applied. These models are then translated into a concrete serialization format and wrote to file objects created on the local file system of the mobile device using a `FileOutputStream`[19]. When all the data contained in a replica had been stored, the file object was closed and removed from the local file system (a device's internal memory) to avoid influences on consecutive benchmarks. The conceptual workflow of the storage and serialization benchmark is described in Algorithm 12:

Algorithm 12: Storing data replicas

Set $Formats \leftarrow$ IdentifySupportedSerializationFormats($\langle f_1, \ldots, f_l \rangle$) ;
foreach $f_i \in Formats$ **do**
  Set *Replicas* $\leftarrow$ CopyDataChunksToInternalMemory($\langle r_1^{f_i}, \ldots, r_m^{f_i} \rangle$) ;
  foreach $r_j^{f_i} \in$ *Replicas* **do**
    LoadReplicaToInternalMemory($r_j^{f_i}$) ;
    while *current run* < *number of iterations* + *1* **do**
      Set *model* $\leftarrow$ ParseDataReplica($r_j^{f_i}$) ;
      Set $t_0 \leftarrow$ `System.getCurrentTimeMillis()` ;
      Set *file* $\leftarrow$ CreateFileObjectOnLocalFileSystem() ;
      WriteModelToLocalFileSystem(*file*, SerializeInMemoryModel(*model*, f_i)) ;
      CloseLocalFileObject(*file*) ;
      Set $t_1 \leftarrow$ `System.getCurrentTimeMillis()` ;
      WriteResultsToLogfile($t_1 - t_0$, *file.getSize()*, *model.countTriples()*, f_i) ;
      RemoveFileFromFileSystem(*file*) ;
      CleanUp $\rightarrow$ InitiateGarbageCollector() ;
    end
    RemoveReplicaFromInternalMemory($r_j^{f_i}$) ;
  end
end

The results of the storage and serialization benchmark compartmentalized by framework and serialization format are depicted in Figure 6.6, 6.7, and 6.8. The plotted graphs represent the storage performance measured in processed triples per second ratios for writing data replicas of particular sizes to file objects on the local filesystem. The size of the data replicas is marked on

[19] Javadoc of the `FileOutputStream` class: `http://download.oracle.com/javase/1.4.2/docs/api/java/io/FileOutputStream.html`

the x-axis; the processed triples per second ratios are displayed along the y-axis. In addition, we also included an illustration of the storage performance compartmentalized by device at the end of this section, which is depicted in Figure 6.9.

6.3.2.1 Androjena

Since Androjena supports the three most popular RDF serialization formats, we measured the storage performance for each format separately and compared them against each other to see whether the differences in parsing performance are also visible for writing RDF replicas to the local file system of a mobile device. Additionally, we compared the storage runtimes of each format to see which format yields the fastest storage times. We were able to process data replicas containing 2,000 triples on the HTC G1, 20,000 triples on the Motorola Milestone, 50,000 triples on the Samsung Galaxy S I9000, and 100,000 triples on the Dell Streak depending on the serialization format. As a result, the size of the data replicas that can be serialized and stored using the Androjena framework correlates with the amount of physical main memory installed on a device.

Androjena yields best storage performance when RDF data are serialized using the N3 format, followed by RDF/XML and N-Triples (see Figure 6.6). The cumulated average storage performance per serialization format across all replicas and devices amounts to 630.34 triples per second using the N3 format, 348.53 triples per second using the RDF/XML format, and 169.05 triples per second using the N-Triples format. Looking specifically at the storage performance per serialization format and device, the average storage performance in terms of processed triples per second for the N-Triples format accounts to 50.82 triples per second on the HTC G1 (see Table A.5), 136.81 triples per second on the Motorola Milestone (see Table A.5), 248.62 triples per second on the Samsung Galaxy S I9000 (see Table A.5), and 239.93 triples per second on the Dell Streak (see Table A.5). Using the RDF/XML serialization format, average serialization and storage performance grows by the factor 1.71 on the HTC G1, by the factor 2.26 on the Motorola Milestone, by the factor 2.22 on the Samsung Galaxy S I9000, and by the factor 1.86 on the Dell Streak. Storing data replicas serialized in N3 format, triples per second ratios grows by the factor 2.07 on the HTC G1, by the factor 2.03 on the Motorola Milestone, by the factor 1.82 on the Samsung Galaxy S I9000, and by the factor 1.59 on the Dell Streak.

As these numbers indicate, the slowest storage performance was measurable for N-Triples-serialized data. The average triples per second ratios of the N-Triples format for all devices and replicas compared to the RDF/XML and the N3 format were approximately 2 times respectively 4 times lower. Hence, we can observe that triples per second ratios almost double from one serialization format to another. As obvious, the growth in terms of triples per second performance from RDF/XML to N3-serialized data is not that distinctive on the two more powerful devices. Additionally, the lowest storage performance was measured on the HTC G1 across all serialization formats.

The storage performance of Androjena is significantly slower for smaller data replicas irrespectively of the serialization format; for instance, triples per second ratios almost grow by the factor 7.18 on the Samsung Galaxy S I9000 when replicas containing 500 triples instead of 10 triples are serialized in RDF/XML format (see Figure 6.6). However, the increase in triples per second performance is observable on all devices and formats. As a consequence, for comparably small data replicas whose sizes range between 10 and 200 triples, we could notice a constant increase in storage performance on all devices and for all serialization formats. The highest storage performance could be measured for data replicas whose size lies between 200 and 2,000 triples. We

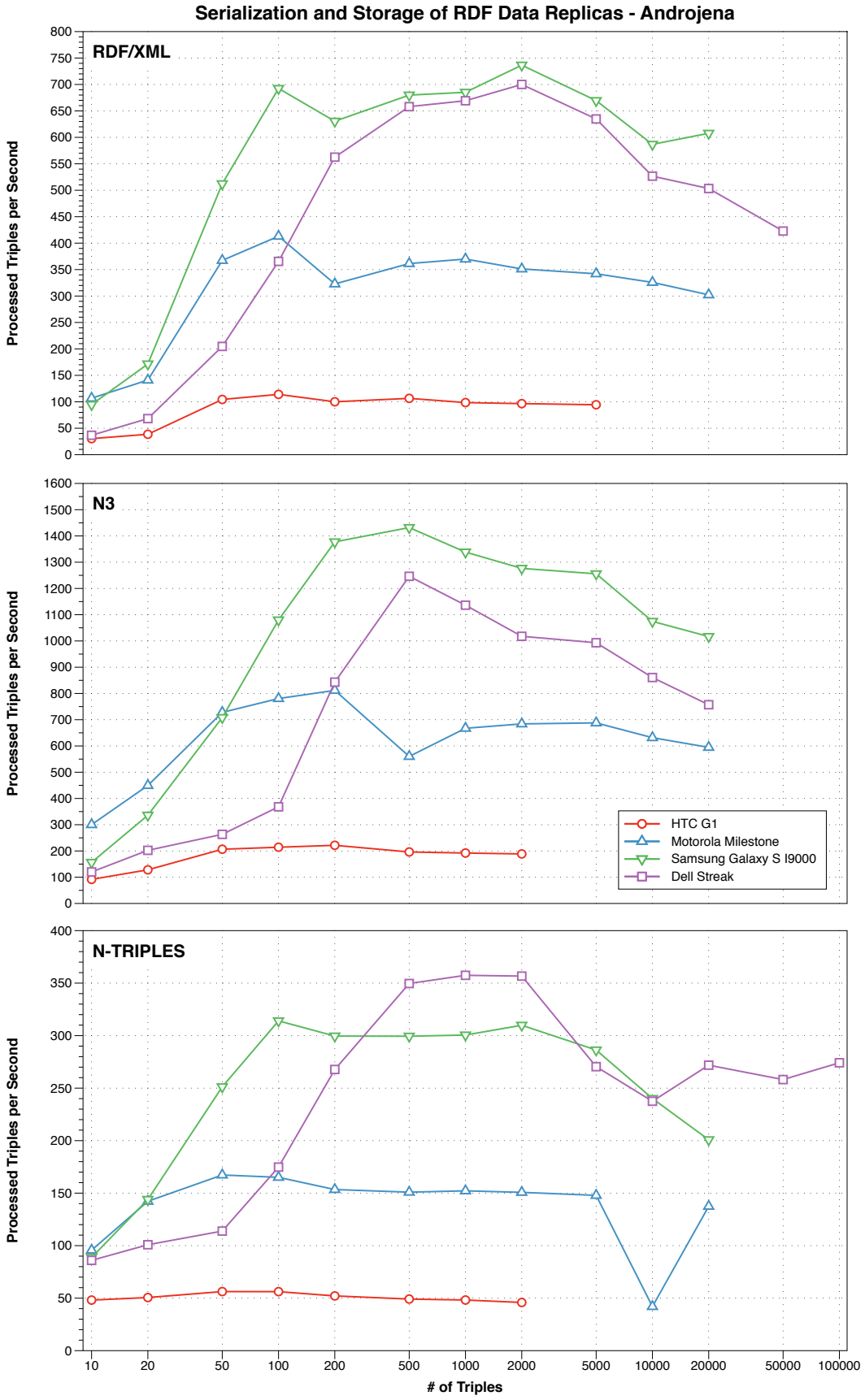


FIGURE 6.6: Serialization and storage performance of the Androjena framework separated by serialization format

believe that the reasons for this runtime behavior can be found in the efforts related to the creation and closing of file objects on the local filesystem.

The storage performance in terms of triples per second ratios begins to decrease again for larger data models with more than 500 to 2,000 triples depending on the device. The slowest storage performance was measurable on the HTC G1 where triples per second ratios across all serialization formats were approximately 3.38 times lower compared to the Motorola Milestone and 5.68 times lower compared to the Samsung Galaxy S I9000. Furthermore, the Samsung Galaxy stores RDF models 1.68 times faster than the Motorola Milestone and 1.29 times faster than the Dell Streak on average. However, we observed a greater variance in processed triples per second ratios on the two more powerful devices, which was not that distinctive on the HTC G1 and the Motorola Milestone where processed triples per second ratios remain relatively constant (with a few exceptions – see Figure 6.6). Additionally, we could see that the storage and serialization performance of the Androjena framework scales relatively well with available processor power as well as with available physical main memory.

In summary, Androjena's saving performance scales reasonably well w.r.t. available processing power where best results could be achieved on the Samsung Galaxy S I9000: total storage times were almost six times faster compared to those measured on the HTC G1 for all serialization formats and data replicas. Serializing RDF graphs in the N3 format yields the best triples per second ratios, followed by RDF/XML and N-Triple. The best storage performance results irrespectively of the serialization format could be measured with graphs of sizes between 100 and 2,000 triples depending on the device.

6.3.2.2 μJena

The size of data replicas that can be serialized and stored using the μJena framework is significantly lower compared to data replicas stored using one of the other RDF frameworks. As a consequence, we were not able to process data replicas with more than 5,000 triples on any device. Moreover, on the Motorola Milestone data replicas have to be reduced to 1,000 triples to be processable using the μJena framework despite its larger main memory compared to the HTC G1, which was able to process data replicas containing 2,000 triples. The storage benchmark was conducted with data replicas serialized in N-Triples format since this is the only format supported by μJena (cf. Section 3.3.2). Although by far the least competitive framework in terms of creation and parsing performance, μJena yields the best average storage performance on the Motorola Milestone and the Dell Streak (see Figure 6.9).

As visible on the Androjena framework, storage performance grows with increasing replica sizes; the slowest storage performance in terms of triples per second ratios was measurable for very small replicas containing 10 to 20 triples. Processing performance constantly grows for data replicas the size of which lies in the range of 10 to 200 triples. For data replicas that exceed this size, triples per second performance begins to drop and start to raise again for models containing 1,000 or more triples depending on the device (see Figure 6.7).

Unsurprisingly, the lowest storage performance was measurable on the HTC G1 where processed triples per second ratios range between 282.49 triples/sec. and 428.27 triples/sec. and cumulate to 383.14 triples/sec. in average (see Table A.5). However, the μJena framework shows the least variance in processed triples per second ratios over all data replicas and devices (see Figure 6.9). Processed triples per second ratios range between 316.46 triples/sec. and 811.03 triples/sec. on the Samsung Galaxy S I9000 and 390.63 triples/sec. to 1,050.42 triples/sec. on the Dell

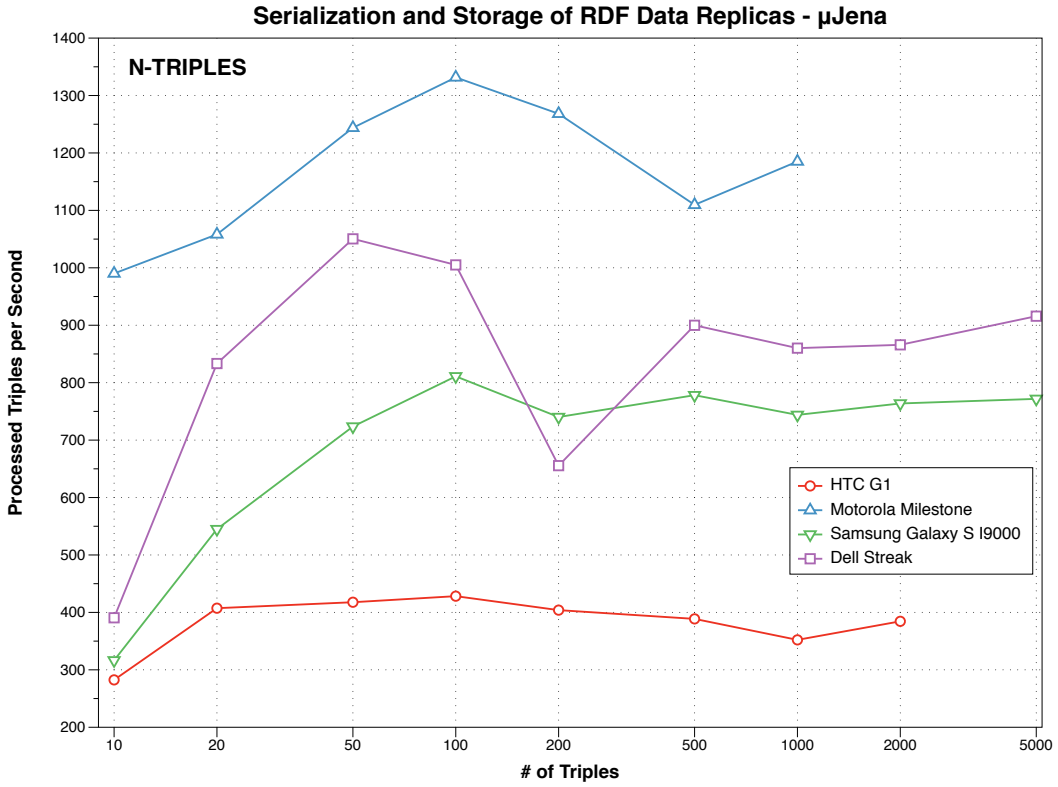


FIGURE 6.7: Serialization and storage performance using the μJena framework

Streak, resulting in an average ratio of 688.21 triples/sec. and 830.74 triples/sec. respectively (see Table A.7 and A.8).

However, the highest storage performance could be measured on one of the less powerful devices in terms of processor speed and memory capacity, the Motorola Milestone, where processed triples per second ratios range between 990.10 and 1,331.56. The average triples per second ratios across all data replicas amount to 1,169.52 triples per second thus being 1.70 times higher than the numbers measured on the Samsung Galaxy S I9000 and 1.41 times higher than those numbers obtained from the Dell Streak (see Table A.6). The average triples per second ratios across all data replicas of the remaining devices account to 383.14 on the HTC G1, 688.21 on the Samsung Galaxy S I9000, and 830.74 on the Dell Streak. As visible from these results, the storage and serialization performance of data replicas using the μJena framework does not directly correlate with available processor speed. Additionally, the variance in processed triples per second ratios is less distinctive on the devices from the lower-market segment.

6.3.2.3 Mobile RDF

The storage and serialization benchmark has been conducted with data replicas serialized in RDF/XML format since this is the one and only format supported by the Mobile RDF framework. Furthermore, the Mobile RDF framework allows to process data replicas containing 10,000 triples on the HTC G1 and larger replicas containing 20,000 triples on the other devices. Although the Samsung Galaxy S I9000 and the Dell Streak incorporate 512 MB of physical main memory compared to 256 MB built into the Motorola Milestone, it was not possible to process data replicas beyond the margin of 20,000 triples. Hence, a doubling of physical main memory from

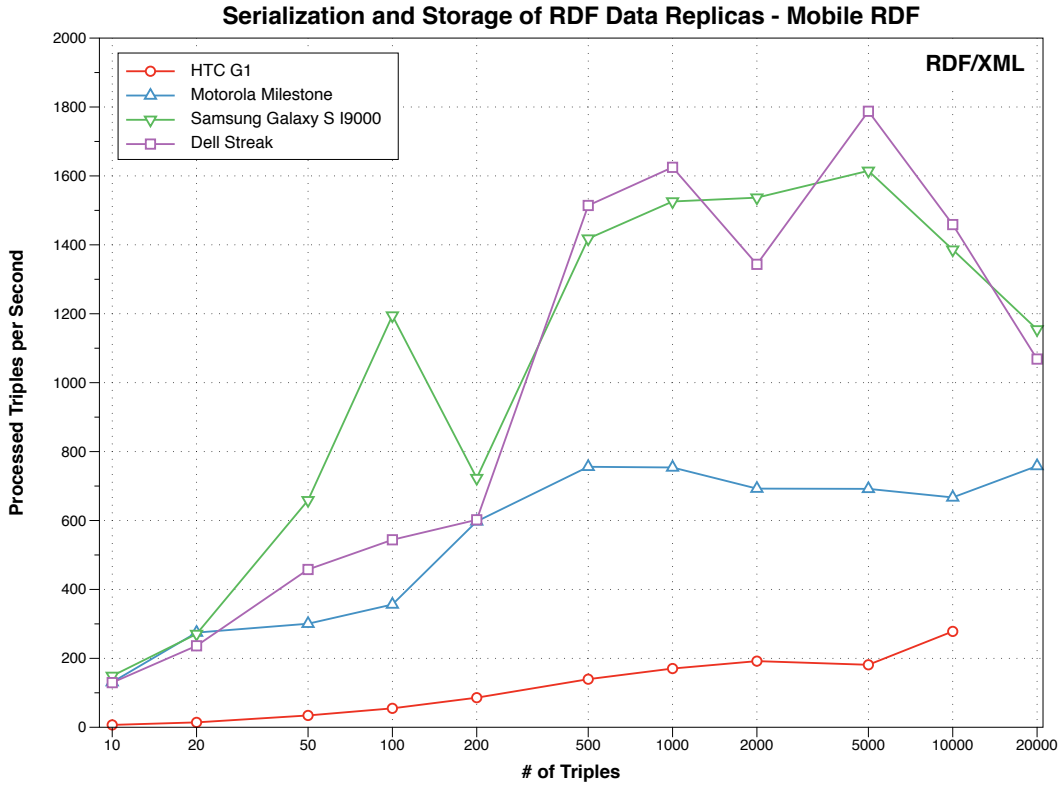


FIGURE 6.8: Serialization and storage performance using the Mobile RDF framework

256 MB to 512 MB, as opposed to the increase from 192 MB to 256 MB, does not result in the capability to process larger data replicas.

As visible on the other two frameworks, the triples per second performance ratios increase with growing data replica' sizes. The slowest storage performance in terms of triples per second ratios was measurable for the smallest data replica sizes. The storage performance constantly grows for data replicas the size of which ranges between 10 and 500 or 1,000 triples depending on the device. The highest triples per second ratios were measurable for data replicas that contain between 500 and 5,000 triples (see Figure 6.8). For data replicas that exceed this triple range, storage performance considerably drops. However, this behavior was observable on the two upper-level devices only, whereas on the Motorola Milestone, triples per second ratios remain constant for data replicas containing 10,000 or 20,000 triples. Additionally, a higher variance in terms of processed triples per second was observable on the two upper-level devices; this variance was not that distinctive on the two less powerful devices. On the HTC G1, for instance, we could notice an almost constant increase in storage performance with growing data replicas.

Mobile RDF scales reasonably well with available processing power where best results could be achieved on the Samsung Galaxy S I9000 and the Dell Streak (see Table A.7 and A.8); total storage times were 9.14 times faster on the Samsung Galaxy compared to those measured on the HTC G1. Although the Samsung Galaxy shows the highest average triples per second ratio, the highest absolute ratios were measured on the Dell Streak. In summary, the average triples per second ratios across all data replicas account to 115.74 triples/sec. on the HTC G1 (see Table A.5), 543.54 triples/sec. on the Motorola Milestone (see Table A.6), 978.87 triples/sec. on the Dell Streak, and 1,057.48 triples/sec. on the Samsung Galaxy S I9000.

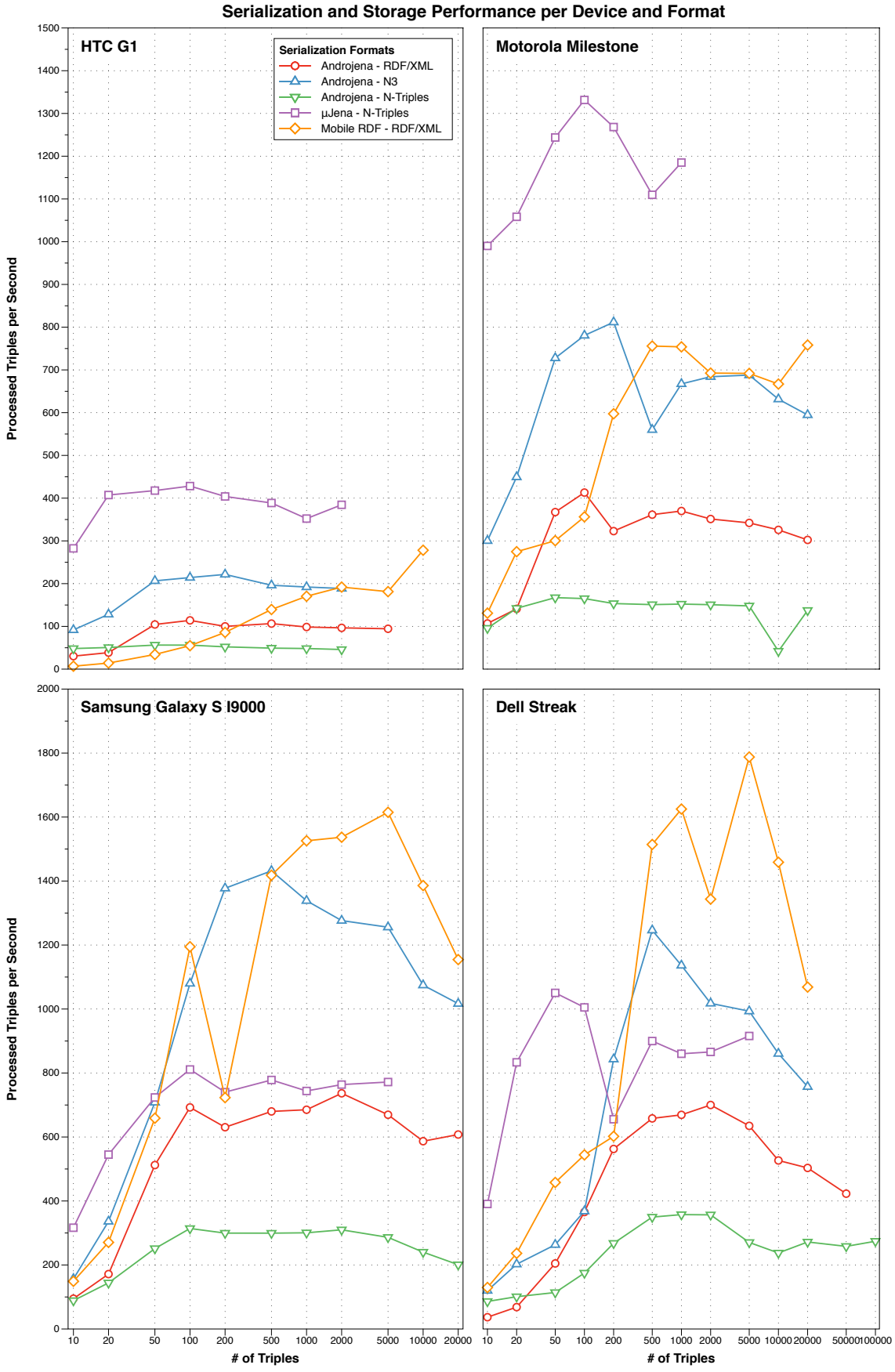


FIGURE 6.9: Serialization and storage performance compartmentalized by device and serialization format

	Androjena			**MicroJena**	**Mobile RDF**			
↓ # of Triples Format →	RDF/XML	N3	N-Triples	N-Triples	RDF/XML[1]	RDF/XML[2]	RDF/XML[3]	RDF/XML[4]
10	1.847	1.385	1.232	1.228	2.678	2.682	2.672	2.678
20	3.472	2.763	2.456	2.448	4.645	4.649	4.639	4.645
50	3.912	4.597	5.804	5.789	6.253	6.257	6.247	6.253
100	7.395	9.243	11.910	11.864	11.864	11.868	11.858	11.864
200	14.411	18.340	23.613	23.521	22.905	22.909	22.899	22.905
500	36.039	46.066	59.775	59.473	56.469	56.473	56.463	56.469
1.000	71.760	91.909	119.616	119.060	111.862	111.866	111.856	111.862
2.000	142.261	182.734	237.096	290.670	221.991	221.995	221.985	221.991
5.000	369.764	459.150	593.851	731.670	555.956	555.960	555.950	555.956
10.000	738.095	917.167	1.187.884		1.110.229	1.110.233	1.110.223	1.110.229
20.000	1.412.416	1.832.181	2.379.020			2.217.582	2.217.572	2.217.578
50.000	3.542.782		5.929.186					
100.000			11.843.279					
					[1] HTC G1	[2] Milestone	[3] Galaxy	[4] Dell Streak

TABLE 6.2: File sizes in bytes of locally stored RDF data replicas depending on RDF framework and serialization format

6.3.2.4 File Sizes of RDF Data Replicas Serialization Sizes

As visible from Figure 3.1 on page 71, the Androjena framework allows to store RDF graphs in three different serialization formats: RDF/XML, N3, and N-Triple. The smallest file sizes had been measured for data replicas being serialized in RDF/XML format, followed by N3, and the N-Triples format (cf. Table 6.2 and Figure 6.10). Surprisingly, the file sizes of N3-serialized data replicas are larger than those serialized with the RDF/XML format although N3 was defined as a more compact and human-readable format [BL06b] that offers a number of abbreviations and short-hand notations for RDF data (cf. [N3P]). The reasons for that fact might be found in a probably immature implementation of the N3 serialization algorithm. On the other hand, Androjena's N3 implementation yields the best storage performance in terms of processed triples per second ratios on all devices. As expected, the largest file sizes were measured for the N-Triples format that also exhibits the slowest storage performance over all serialization formats using the Androjena framework (cf. Figure 6.6). Additionally, no variations in terms of file sizes could be observed among devices, i.e., each replica stored in a particular serialization format exhibits the same file size in bytes on all devices.

The μJena framework allows to store RDF graphs in N-Triples format only, where the file size of RDF data replicas containing between 10 and 1,000 triples is slightly smaller compared to N-Triples serialized data replicas using the Androjena framework. However, the file sizes substantially grow for data replicas the size of which lies between 2,000 and 5,000 triples compared to Androjena. Additionally, μJena also shows no variations in file sizes among different devices.

The serialization algorithm implemented in Mobile RDF, in contrast, produces slightly different file sizes on the included devices for identical data replicas (see Table 6.2). Although Mobile RDF only supports the RDF/XML format, the data replicas stored using the Mobile RDF framework exhibit a significantly higher file size compared to the RDF/XML-serialized data replicas stored with the Androjena framework (see Figure 6.10).

The file sizes of the data replicas in bytes measured for each framework and supported serialization format are depicted in Figure 6.10. As results indicate, although the storage performance of RDF/XML-serialized data of the Mobile RDF framework is higher compared to Androjena, the

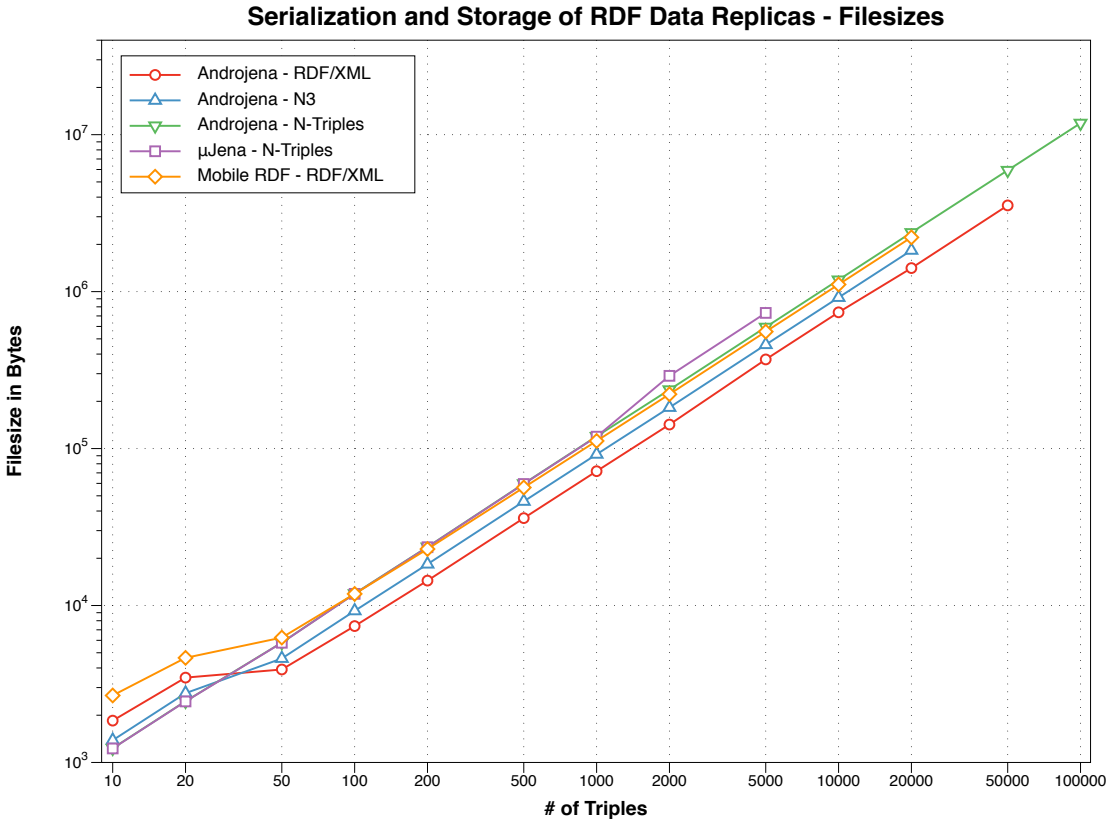


FIGURE 6.10: File sizes in bytes of RDF data replicas stored on the local file system depending on the serialization formats supported by involved RDF frameworks

file sizes of data replicas being stored using the Androjena framework are significantly smaller, in particular for data replicas containing 20,000 and more triples (up to 40% for identical replicas).

6.3.2.5 Summary

As Androjena is the one and only framework to date that supports multiple serialization formats, best performance results had been measured with N3-serialized data replicas, which offer the second best triples per file size ratio, i.e, amount of contained triples per file size in bytes. The best triples per file size ratio was measured for data replicas being serialized in the RDF/XML format. N-Triple serializations not only yields slowest storage performance, they also produce the largest file sizes. Storage times of all frameworks are relatively linear with the amount of triples to be stored, i.e, we could observe a linear scaling between storage runtimes and the amount of triples to be saved on all three frameworks and on each device. However, no significant difference w.r.t. the file sizes between the different frameworks and serialization formats could be found, which indicates that storage algorithms do not make use of, e.g., *QNames*. File sizes of the serialized data replicas are rather similar among all frameworks and devices. Although μJena lies considerably behind the other two frameworks in terms of parsing performance (cf. Section 6.3.1.2), it achieves the highest storage performance over all frameworks on the HTC G1 and the Motorola Milestone as well as in total. The overall storage performance amounts to 767.90 triples/sec. compared to 673.91 triples/sec. and 630.34 triples/sec. measured on the Dell Streak and the Samsung Galaxy S I9000. However, the highest absolute values were measured

on the Dell Streak using the Mobile RDF framework the performance of which is almost similar to the Androjena framework.

6.3.3 Adding Data to RDF Data Replicas

In the course of the insertion benchmark, we measured the total amount of time needed to add a distinct number of triples to RDF data replicas of varying sizes[20]. The data replicas had been copied from the SD card to the internal memory of a device prior to the execution of the insertion benchmark. The data replicas are then parsed and transformed into workable in-memory RDF models to which triples of distinct sizes are added. For each combination of framework and device, we measured the total amount of time needed to insert data sets containing 1, 10, 100, 500, and 1,000 triples. However, a distinction according to different serialization formats was not regarded since all measured operations were performed to in-memory RDF models exclusively wherefore concrete serialization formats are negligible. Algorithm 13 provides an overview of all the relevant steps performed in the course of the insertion benchmark[21]. To distinguish RDF data replicas from RDF models containing the triples to be inserted in the data replicas, such models are denoted as '*data sets*' or '*RDF data models*' throughout this section.

Algorithm 13: Inserting triples into an in-memory RDF graph

```
Set Datasets ← ⟨d_1, ..., d_5⟩ ;
Set d_1 ← 1 triple ;
Set d_2 ← 10 triples ;
Set d_3 ← 100 triples ;
Set d_4 ← 500 triples ;
Set d_5 ← 1000 triples ;
Set Replicas ← CopyDataChunksToInternalMemory(⟨r_1, ..., r_m⟩) ;
foreach replica r_i ∈ Replicas do
    foreach dataset d_j ∈ Datasets do
        while current run < number of iterations + 1 do
            Set model ← ParseDataReplica(LoadDataReplica(r_i)) ;
            Set t_0 ← System.getCurrentTimeMillis() ;
            InsertDataset(d_j → model) ;
            AssertTrue((model.size()_t0 + d_j.size()) == model.size()_t1) ;
            Set t_1 ← System.getCurrentTimeMillis() ;
            WriteResultsToLogfile(t_1 − t_0, model.countTriples(), d_j.size()) ;
            RemoveReplicaFromMainMemory(r_i) ;
            CleanUp → InitiateGarbageCollector() ;
        end
    end
end
RemoveReplicasFromInternalMemory(⟨r_1, ..., r_m⟩) ;
```

The results of the insertion benchmark compartmentalized by framework, device, and data set size are depicted in Figure 6.11, 6.12, and 6.13. The plotted graphs represent the insertion performance measured in processed triples per second ratios for adding a particular number of triples to data replicas of varying sizes. The size of the data replicas is marked on the x-axis; the processed triples per second ratios are displayed both linearly and logarithmically along the

[20] During the insertion benchmark, a strange behavior was observable: while some benchmarks finished successfully for rather large replicas with more than 20,000 triples, other tests failed, probably due to variances in running system and user processes that consume different amounts of main memory.

[21] For readability issues, we omitted initialization as well as detailed logging operations.

y-axis, depending on the framework. Additionally, we provide statistics related to the insertion performance separated by data set size and framework for each device at the end of this section. These statistics are depicted in Figure 6.14, 6.15, 6.16, and 6.17.

6.3.3.1 Androjena

Using the Androjena framework, we were able to add triples to data replicas containing a maximum of 5,000 triples on the HTC G1, 20,000 triples on the Motorola Milestone, 50,000 triples on the Samsung Galaxy S I9000, and 100,000 triples on the Dell Streak.

Insertion times for 1 triple range between 1.3 and 1.6 milliseconds on the HTC G1 (cf. Table A.9), and 0.1 to 1.0 milliseconds on the other, more powerful devices for all data replica sizes (cf. Table A.10, A.11, and A.12). Hence, insertion times remain relatively stable on all data replicas for a particular triple size – even for very large data replicas. As a result, insertion operations are not dependent on or influenced by the size of data replicas and are executed with nearly constant triples per second ratios.

Increasing the data set sizes to 10 or 100 triples, we can observe that insertion times also increase nearly linearly by the factor 10 on all devices. In this respect, the amount of triples to be inserted scales almost linearly with the total time needed to complete the insertion operation. For instance, an increase of the triples to be inserted by the factor ten (e.g., from 10 triples to 100 triples) resulted in an increase of the total time needed for completing the insert operation by the factor 8 to 10 in average depending on the device (see Figure 6.11). An identical behavior is observable if the number of triples to be inserted was increased from 100 to 500 triples where insertion times grew by the factor 5.

The average processed triples per one second ratios range between 568.67 triples/sec. and 689.97 triples/sec. on the HTC G1, 2,171.31 triples/sec. and 2,797.98 triples/sec. on the Motorola Milestone, 3,680.56 triples/sec. and 5,706.98 triples/sec. on the Samsung Galaxy S I9000, and 4,093.19 triples/sec. and 6,056.55 triples/sec. on the Dell Streak (see Figure 6.11). Only when large data sets containing 1,000 triples or more are inserted into data replicas of 100,000 triples, a substantial decrease of processed triples per second ratios was observable on the Samsung Galaxy S I9000 and the Dell Streak (cf. Table A.11 and A.12).

However, on the Samsung Galaxy S I9000 we could observe that Androjena yields the best processed triples per second performance when data sets containing several hundred triples are inserted; for instance, adding 100 triples yields a processed triples per second ratio of 5,706.98 in average on all data replicas, which was the highest value among all measurements for larger data sets to be inserted[22] (see Figure 6.16).

On the two low and middle-segment devices, the highest triples per second performance can be observed when inserting very small models containing not more than 10 triples, whereas on the two upper-segment devices, the highest triple per second performance was observable when data sets of 100 respectively 500 triples were inserted.

In total, the best average performance results have been measured on the Dell Streak where the average processed triple per second ratios in total, i.e., among all data replicas and inserted

[22] Although the measured average triples per second ratios for inserting 1 triple are slightly higher on the Dell Streak, we do not elaborate on those numbers here any further due to measuring inaccuracies attributed to the limited precision of Java-based measurement methods for comparably short execution times, so-called *micro benchmarks* [Goe05]. For a discussion related to the problematics of performance benchmarks in general and micro benchmarks in particular see [Goe04, Goe05].

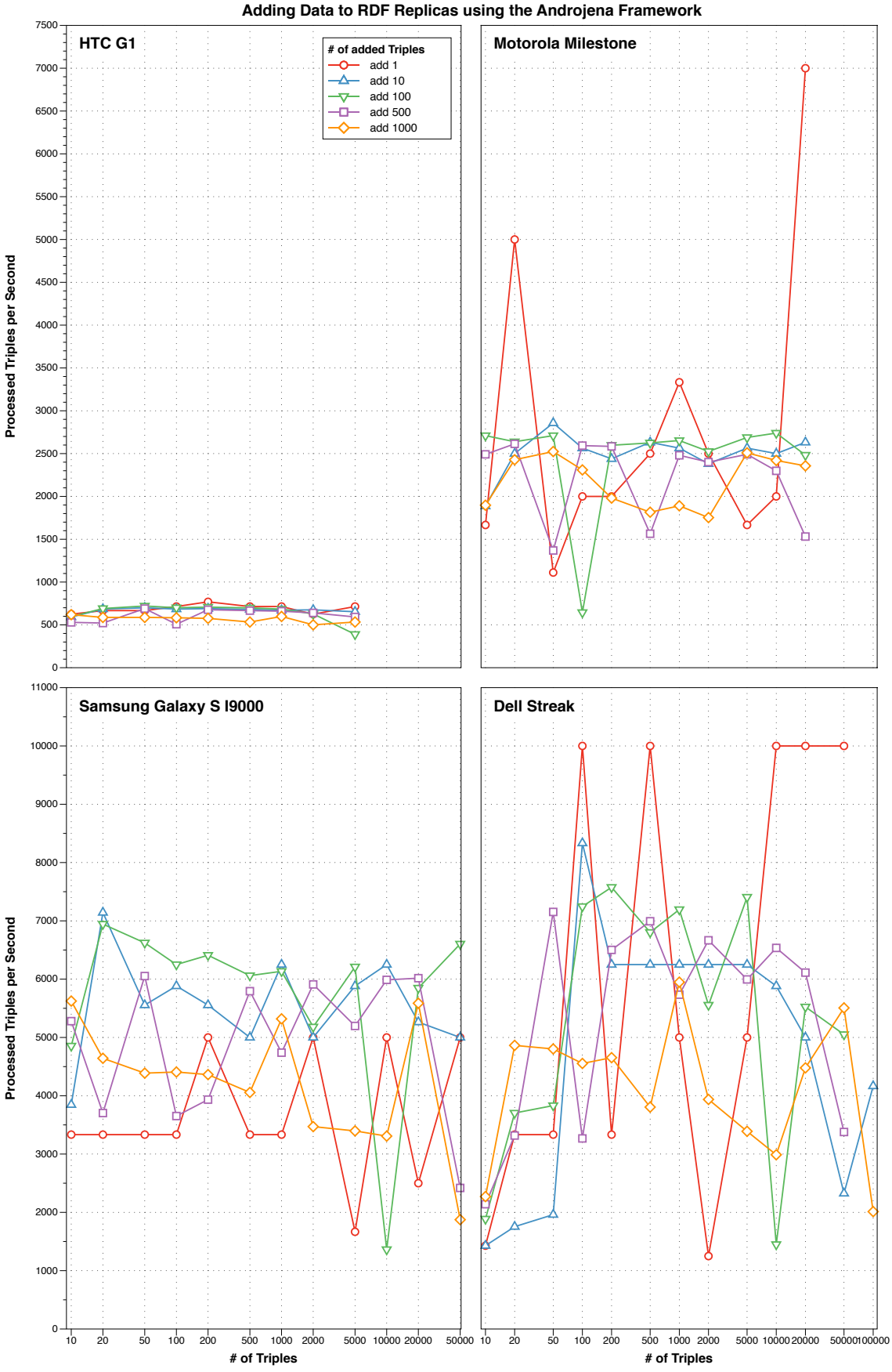


FIGURE 6.11: Performance of insertion operations on included devices using the Androjena framework

data sets, are slightly higher than on the Samsung Galaxy S I9000. In the range of the analyzed insertion data set sizes, we can observe a slight but constant decrease in processed triples per second ratios while the amount of triples to be inserted increases on the HTC G1 and Motorola Milestone.

On the two upper-level devices, we could notice an increased variance among the average processed per triples ratios, which was not that distinctive on the two other devices (see Figure 6.11). However, derivations among the average insertion times can be attributed to different situations and system processes running while the benchmarks were executed. Looking specifically at the HTC G1 and the Motorola Milestone, we can observe that an increase of CPU speed from 350 MHz to 600 MHz (factor 1.72) resulted in a multiplicity of processed triples per second ratio of the factor 3.81 using the Androjena framework. In this respect, the performance multiplies by approximately 2 from the Motorola Milestone to the two upper-segment devices and by the factor 8 from the HTC G1. In total, insertions of a few hundred triples can be performed in a few milliseconds and without noticeable delay on all devices using the Androjena framework, whereas for insertions of larger data sets containing 1,000 triples or more, an upper-level device is necessary to process such insertions within the range of milliseconds.

6.3.3.2 μJena

Just as visible in other benchmark tests, the processing performance of the μJena framework in particular for larger data replicas dramatically drops and this was likewise the case for insertion operations. Additionally, we were not able to process data replicas with more than 5,000 triples on any of the devices due to limited physical main memory and unthrifty management of available system resources probably attributed to unoptimized memory-intensive internal data structures implemented in μJena. Therefore, we have to limit data replica sizes to 2,000 triples on the HTC G1, and 5,000 triples on the Samsung Galaxy S I9000 and the Dell Streak. Although the Motorola Milestone incorporates 256 MB of main memory compared to 192 MB of the HTC G1, it was not possible to scale our tests up to replica sizes of 2,000 triples and above; as a consequence, the insertion benchmark could only be performed with a maximum of 1,000 triples on the Motorola Milestone.

On the HTC G1, execution times grow disproportional for data replicas containing more than 100 triples on all inserted data set sizes (see Table A.13). Average processed triples per second ratios over all data replicas dramatically decrease when models with more than a few hundred triples are to be inserted: they drop from approximately 20 triples per second for data sets containing 100 triples to approximately 6 triples per second for data sets containing 500 triples on the HTC G1 (see Table A.13).

On the Motorola Milestone, the processed triples per second ratios were almost three times higher on average for inserting data sets containing 1 or 10 triples over all data replicas compared to the HTC G1, but assimilate for data sets containing 100, 500, and 1,000 triples. An unusual phenomenon was observable on the Motorola Milestone when data sets containing 100 triples or more are added to replicas containing more than 50 or 100 triples as processed triples per second ratios grow from low single-digit range to low and middle tens range (see Table A.14). This phenomenon was not observable on any other device where processed triples per second ratios decrease with growing replica sizes for all inserted data sets (see Figure 6.12).

μJena performs significantly faster on the Samsung Galaxy S I9000 where the total amount of time needed to add 1,000 triples to data replicas containing 5,000 triples ranges at factors

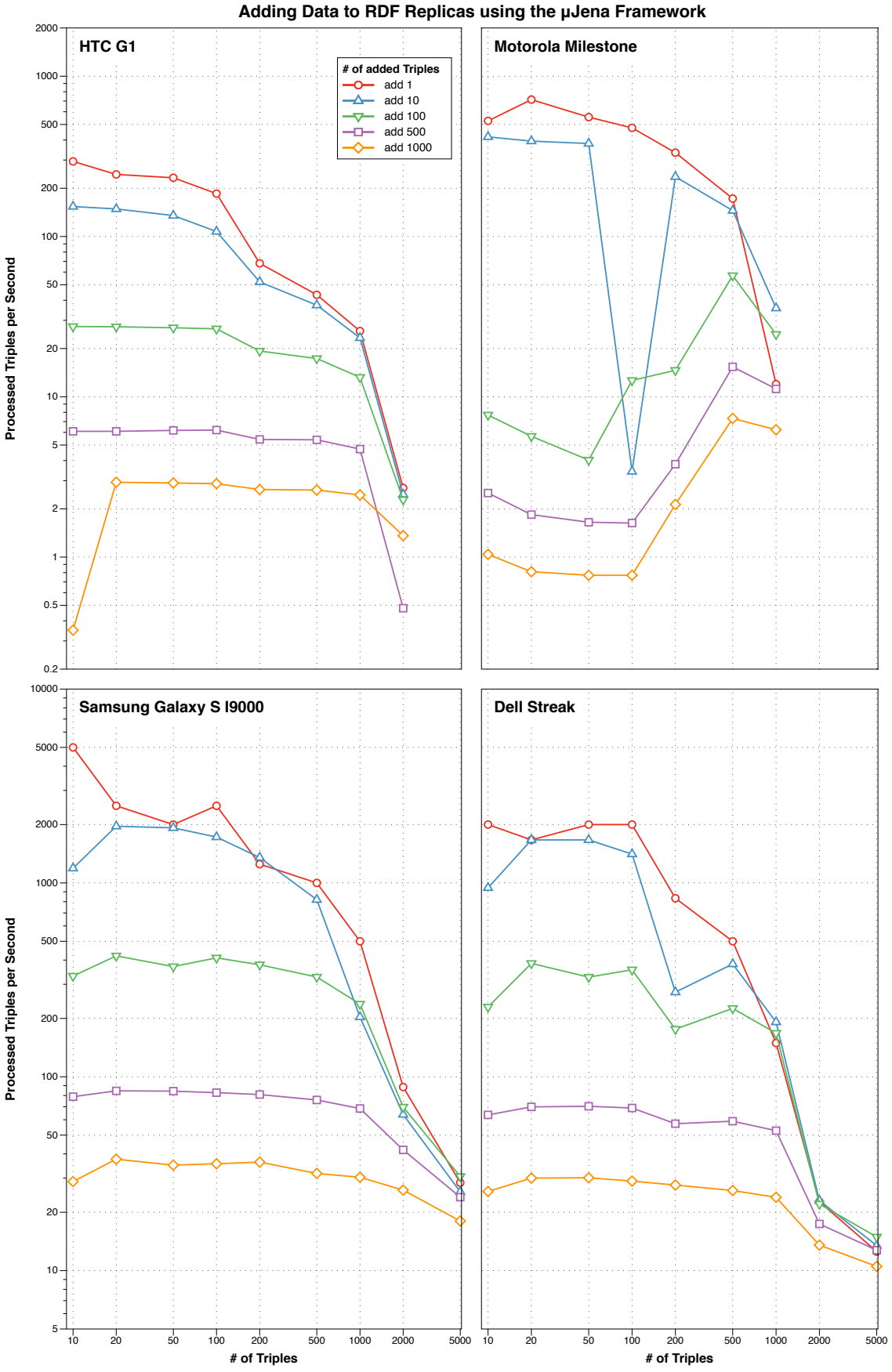


FIGURE 6.12: Performance of insertion operations on included devices using the μJena framework

of around 10 to 20 below the averagely measured times on the HTC G1 and the Motorola Milestone (see Table A.15). For adding rather small data sets containing only a few triples, the processed triples per second ratio grows almost by the factor 12 compared to the HTC G1 and factor 4 compared to the Motorola Milestone. This behavior was also observable on the Dell Streak, although the average processed triples per second ratio was slightly lower compared to the Samsung Galaxy S I9000 (see Table A.16). As obvious from these numbers, insertion operations of the μJena framework benefit from available processing power rather than from available physical main memory.

For models with less than 1,000 triples, insertions of data sets containing less than 100 triples perform almost linear per triple size; this means that there is not a noticeable variance among insertions of small RDF models on all data replicas. If the size of data replicas grows beyond 1,000 triples, insertion times significantly increase for data sets containing more than 100 triples. For instance, if 500 triples are to be inserted in a replica containing 1,000 triples, the insertion operation takes 7.2 seconds in average; if, in contrast, only 100 triples are to be inserted in data replicas of the same size, the insertion operation takes approximately 421.5 milliseconds in average on the Samsung Galaxy S I9000. If the number of triples to be inserted doubles from 500 to 1,000, insertion times increase by the factor 4 to 6 in average on all devices.

In general, small data sets can be added to data replicas very fast, although the total amount of time needed to execute an insertion operation increases disproportional with growing data replica sizes. However, insertion times for adding triples to large data sets are balanced and almost constant, but the overall time needed to perform the insertion operation exceeds a tolerable maximum from a usability perspective and provides results that are unacceptable for real-world replication scenarios (cf. Table A.13 - A.16 and Figure 6.14 - 6.17).

6.3.3.3 Mobile RDF

On the HTC G1 and the Motorola Milestone, we scaled our insertion tests up to data replicas containing 10,000 triples; on the Samsung Galaxy S I9000 and the Dell Streak, we were able to expand data replica sizes to 20,000 triples due to 512 MB main memory compared to 192 MB and 256 MB built into the HTC G1 and the Motorola Milestone.

On the HTC G1 device, the Mobile RDF framework was capable of performing insertion operations applied to data replicas below 10,000 triples in reasonable time (triples per second ratios range between 400 and 1,100 on average – cf. Table A.17); when triples are added to replicas that contain 10,000 triples or more, the performance significantly drops from 981.08 triples per second in average to 1.82 triples per second for inserting 1 triple (see Figure 6.13 and 6.14). Additionally, the same effect was visible for inserting 10 triples into data replicas containing 10,000 triples where processing performance declines from 911.35 triples per second in average to 13.00 triples per second (see graph for adding 10 triples in Figure 6.14). Interestingly, the processed triples per second performance increased with growing data sets sizes for data replicas that contain 10,000 triples (see graphs for adding 100, 500, and 1,000 triples in Figure 6.14). However, this behavior was observable on the HTC G1 and—although less distinctive—on the Samsung Galaxy S I9000 only, although tests were performed with replica sizes of 20,000 triples on the Samsung Galaxy.

On the Motorola Milestone, the insertion performance was approximately 4 times higher compared to the HTC G1 and ranges between 3,866.67 triples per second for inserting 1 triple to a data replica containing 20,000 triples and 2,121.31 triples per second for inserting 1,000

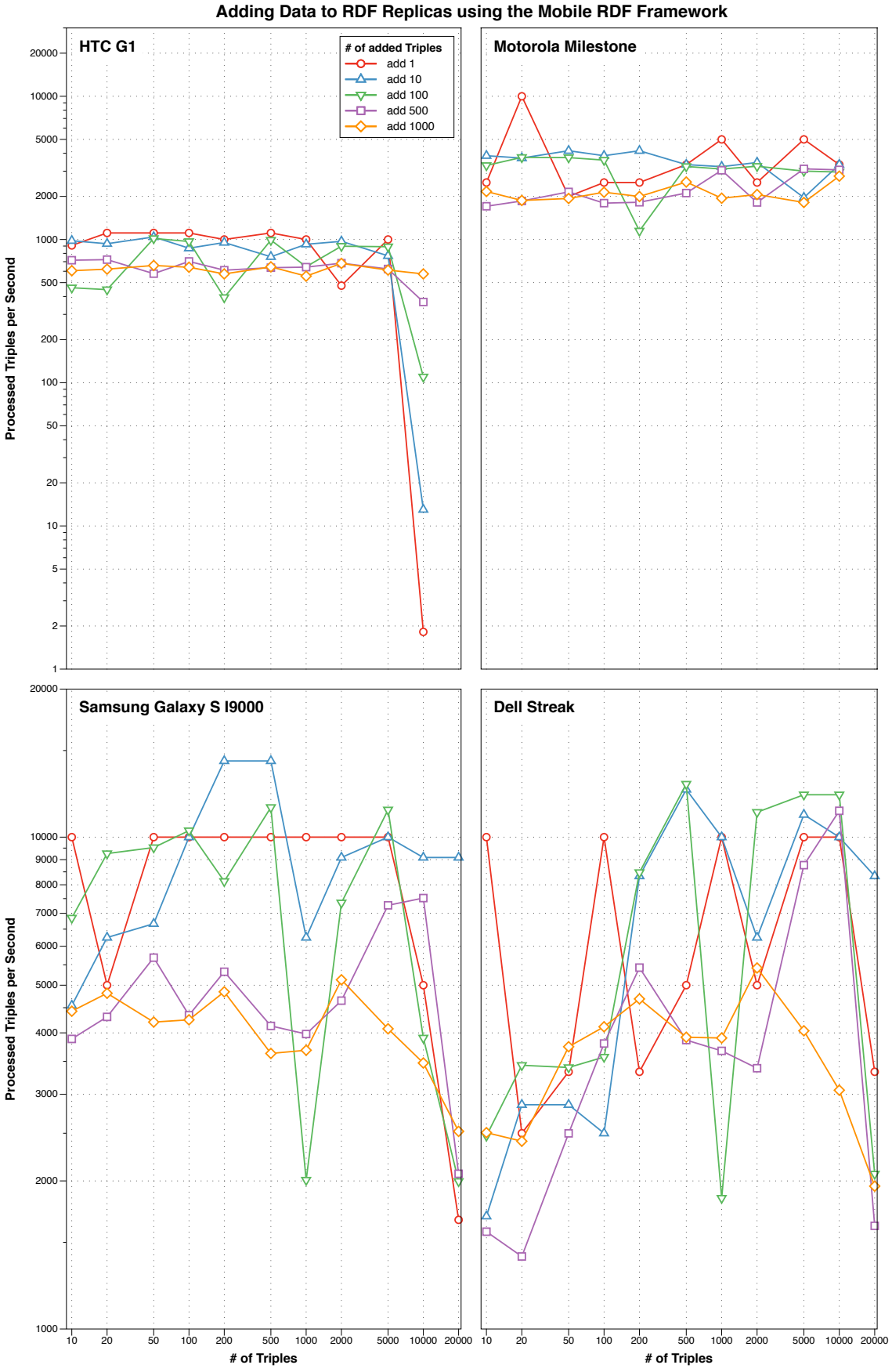


FIGURE 6.13: Performance of insertion operations on included devices using the Mobile RDF framework

triples to a data replica of the same size (see Table A.18). The average processed triples per second ratios of the Samsung Galaxy S I9000 and the Dell Streak over identical data replica and RDF model sizes range between 8,333.33 triples/sec. and 4,097.04 triples/sec. respectively 6,948.82 triples/sec. and 3,614.39 triples/sec. (see Table A.19 and A.20).

However, all devices showed a decrease in processed triples per second performance with growing data set sizes (cf. Figure 6.13). Insertion performance between the smallest data set containing 1 triple and the largest data set containing 1,000 triples differs by the factor 1.59 on the HTC G1, by the factor 1.82 on the Motorola Milestone, by the factor 2.03 on the Samsung Galaxy S I9000, and by the factor 1.92 on the Dell Streak. As obvious from these numbers, the variance in average insertion performance is more distinctive on devices with a higher CPU clock speed. In contrast to the two less powerful device where the highest insertion performance in terms of processed triples per second ratios was achieved for data sets containing 1 triple, the highest insertion performance on the two upper-market segment devices was measured when data sets containing around 10 or 100 triples are inserted into data replicas (see Figure 6.13).

In general, the processed triples per second ratio ranges around 726,26 triples per second on the HTC G1 (cf. Table A.17), 2,968.64 triples per second on the Motorola Milestone (cf. Table A.18), 6,757.34 triples per second on the Samsung Galaxy S I9000 (cf. Table A.19), and 5,631.39 triples per second on the Dell Streak (cf. Table A.20) in average over all data sets and all data replica sizes. With smaller data sets containing less than 500 triples, insertion times almost scale linearly with the amount of triples to be inserted.

As obvious from these numbers, the insertion performance of the Mobile RDF framework scales relatively linear with available processor performance where the best performance results could be measured on the Samsung Galaxy S I9000: the processing performance of insertion operations over all data replicas and all data sets was 1.20 times higher compared to the Dell Streak. Comparing the HTC G1 and the Motorola Milestone, insertion operations are performed by the factor 4.08 faster on the Motorola Milestone than on the HTC G1 in average. On the Samsung Galaxy S I9000 and the Dell Streak, the performance of insertion operations roughly doubles by the factor 2.27 on the Samsung Galaxy and by the factor 1.89 on the Dell Streak compared to the Motorola Milestone.

6.3.3.4 Summary

In general, the insertion performance of all frameworks scales reasonably well with available processing power where the highest insertion performance in terms of processed triples per second ratios over all data replicas, devices, and data sets has been measured using the Mobile RDF framework. A very important aspect regarding the modification of RDF data replicas is that the processing time per inserted data set size remains independent from the size of the data replicas, i.e., the runtime behavior of insertion operations is not influenced by the size of the in-memory RDF graph to which a distinct number of triples are added. This requirement was fulfilled solely by the Androjena and the Mobile RDF framework: processing time per inserted data set remains relatively stable across the different data replicas. Using the μJena framework, we could notice a substantial decrease in insertion performance on both dimensions – with increasing data set sizes as well as with increasing data replica sizes, which renders the μJena framework inappropriate for extensive insertion operations. In summary, insertions of data sets containing several hundreds or thousand triples can only be performed on newer and more powerful devices using the Androjena or Mobile RDF framework in acceptable time; on slower devices, insertion size should not exceed a maximum of 500 triples.

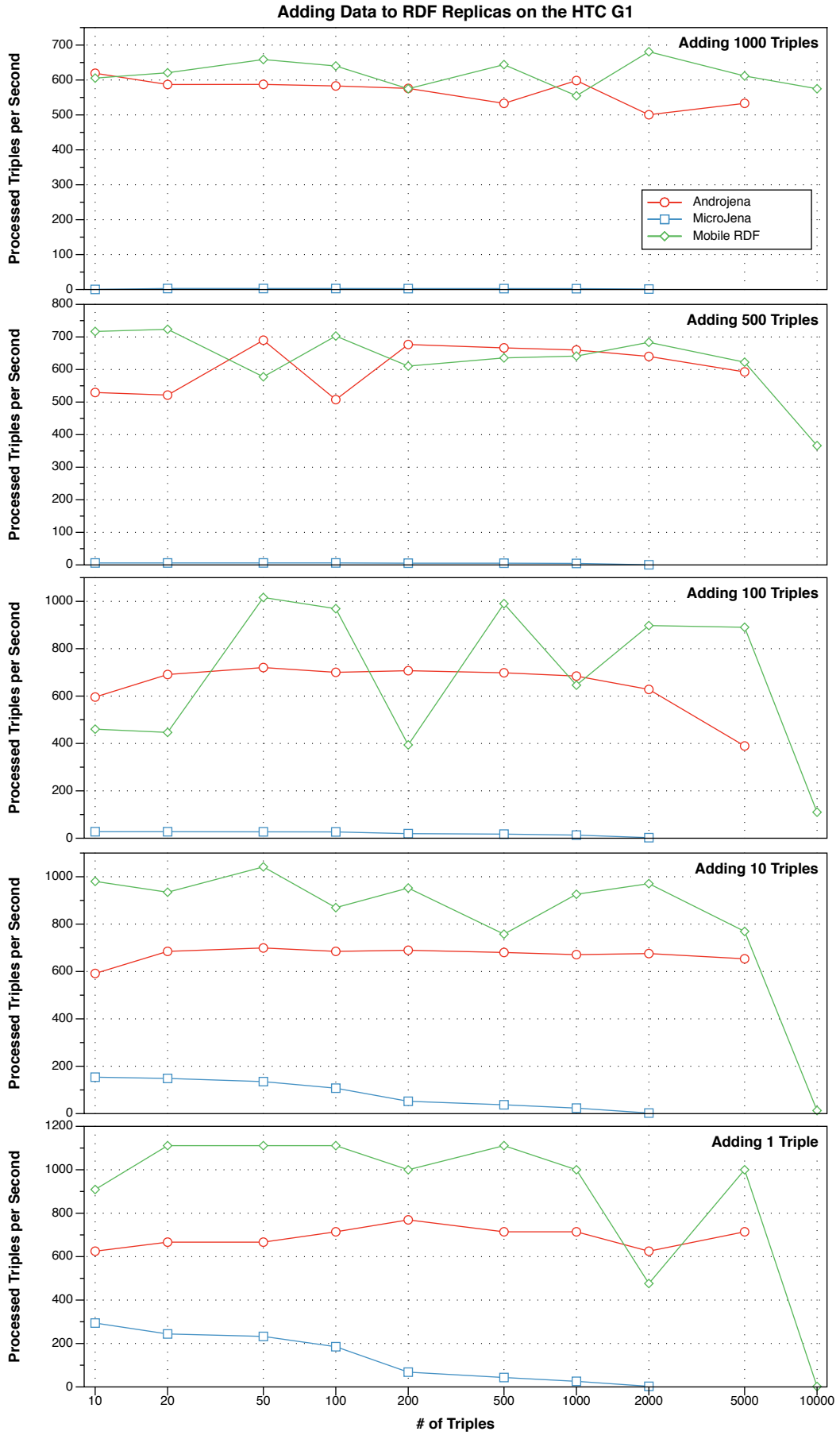


FIGURE 6.14: Performance per framework for adding data sets of specific size to RDF data replicas on the HTC G1

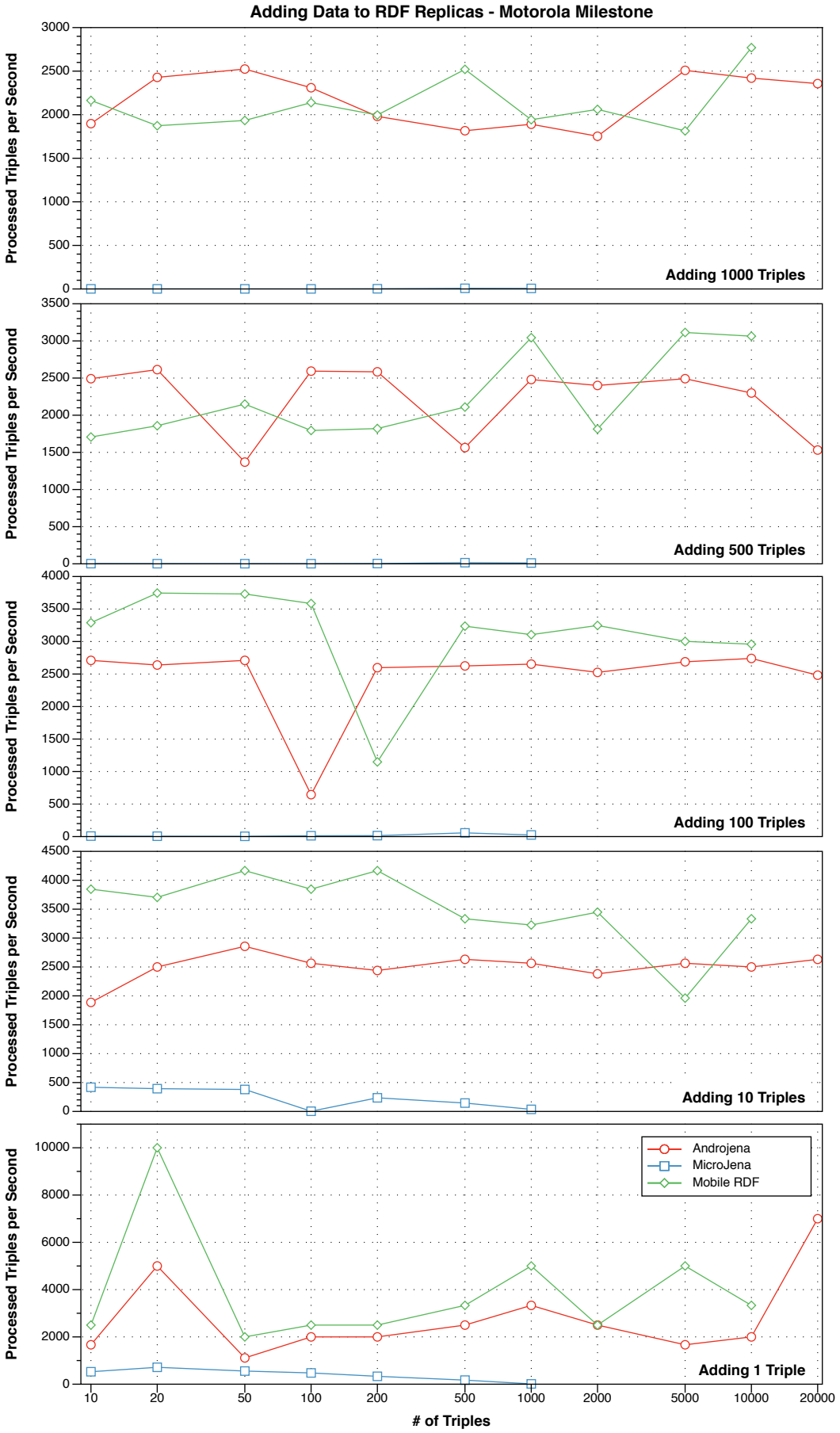


FIGURE 6.15: Performance per framework for adding data sets of specific size to RDF data replicas on the Motorola Milestone

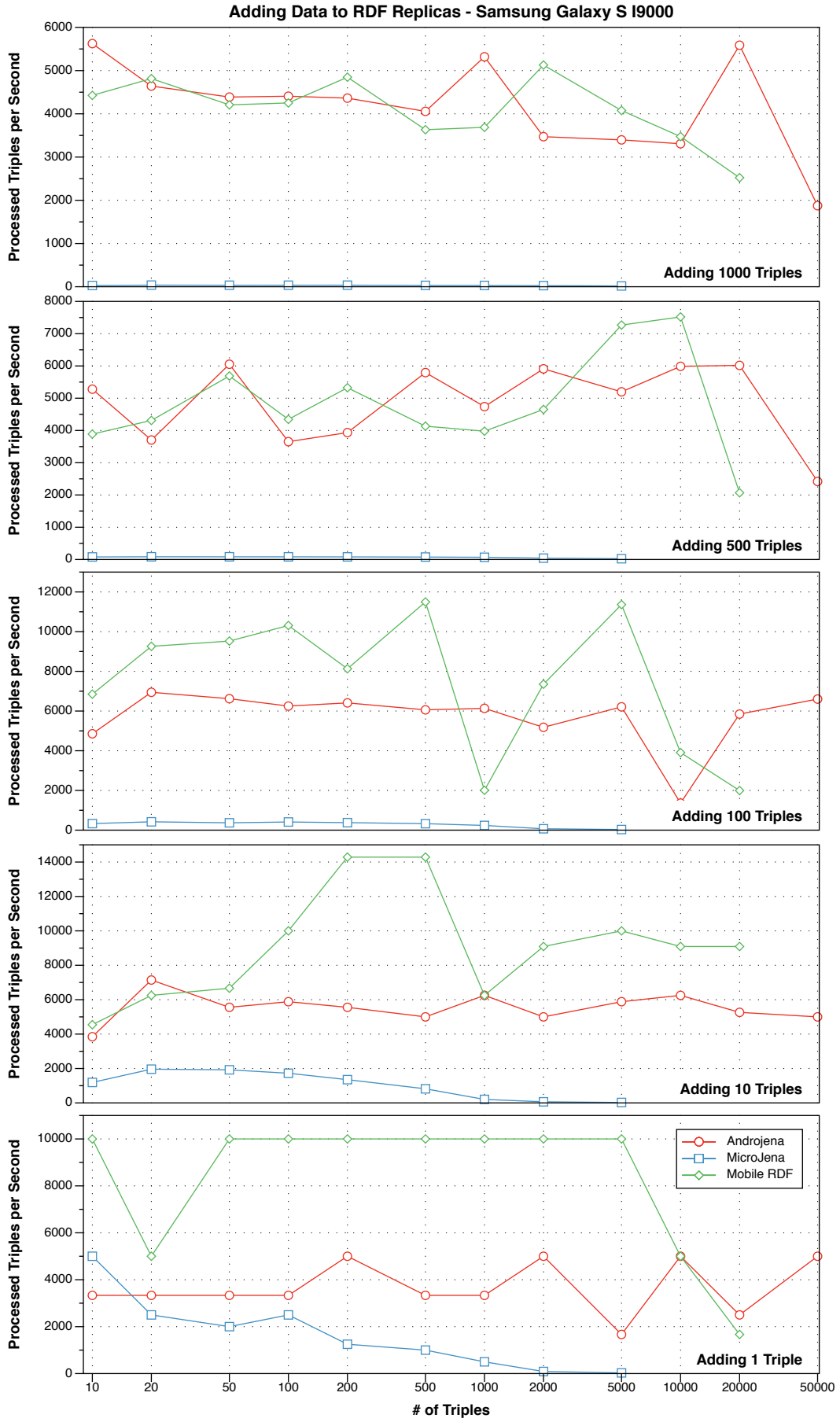


FIGURE 6.16: Performance per framework for adding data sets of specific size to RDF data replicas on the Samsung Galaxy S I9000

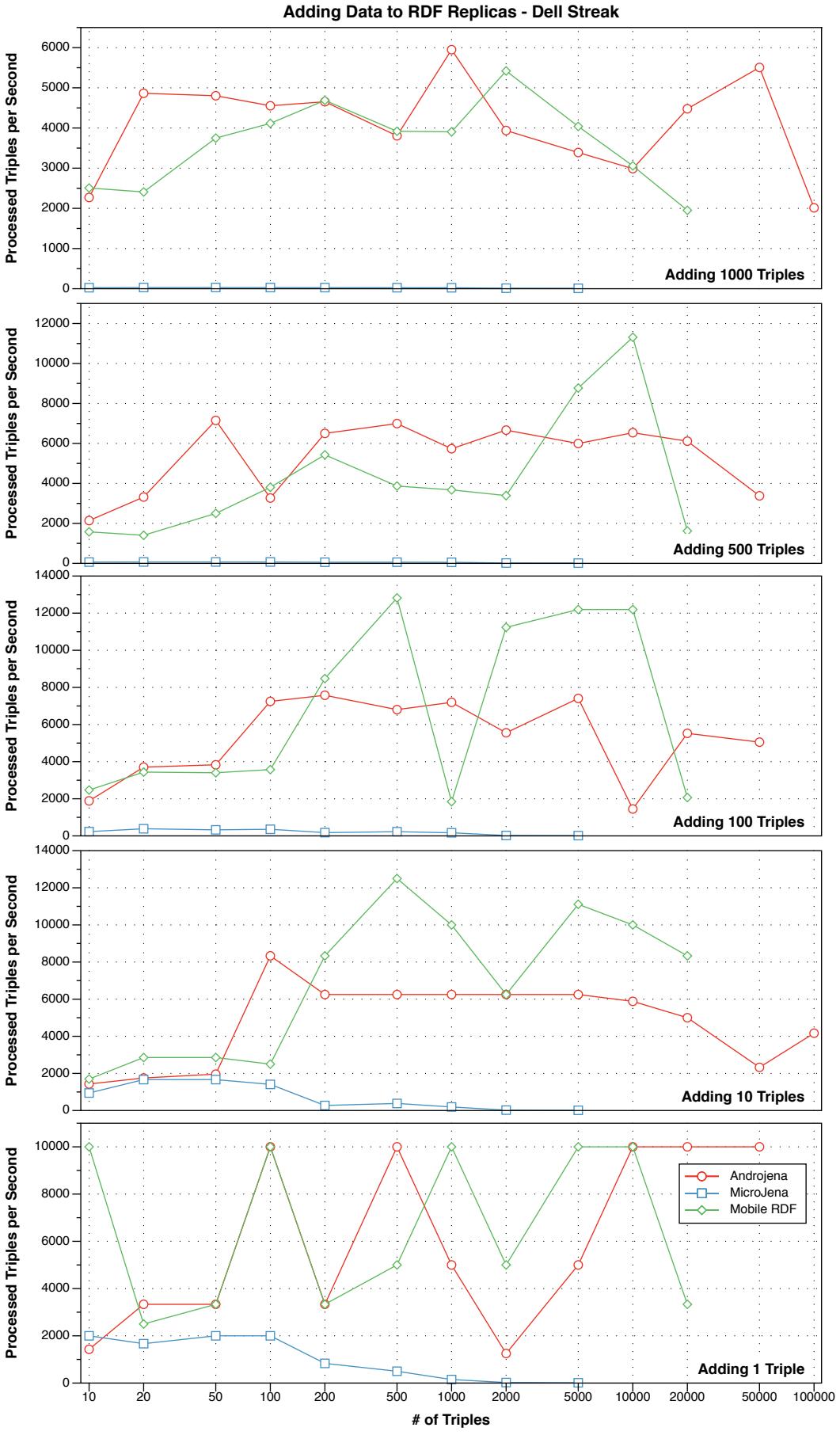


FIGURE 6.17: Performance per framework for adding data sets of specific size to RDF data replicas on the Dell Streak

6.3.4 Removing Data from RDF Data Replicas

In the course of the removal benchmark, we ascertained the removal performance of the involved RDF frameworks. Hence, for each combination of framework and device, we measured the total amount of time needed to remove a distinct number of triples added during the insertion benchmarks from RDF data replicas of varying sizes. Before data is added to and removed from a data replica, it is parsed and transformed into a workable in-memory RDF model to which insertion and removal operations are applied. We used the identical data set sizes as presented in the insertion benchmark for the removal benchmark. Furthermore, a distinction according to different serialization formats was not regarded since all measured operations were performed to in-memory RDF models exclusively wherefore concrete serialization formats are negligible. Algorithm 14 provides an overview of the main steps performed in the course of the removal benchmark[23].

Algorithm 14: Removing inserted triples from an in-memory RDF graph

```
Set Datasets ← ⟨d_1, ..., d_5⟩ ;
Set d_1 ← 1 triple ;
Set d_2 ← 10 triples ;
Set d_3 ← 100 triples ;
Set d_4 ← 500 triples ;
Set d_5 ← 1000 triples ;
Set Replicas ← CopyDataChunksToInternalMemory(⟨r_1, ..., r_m⟩) ;
foreach replica r_i ∈ Replicas do
    foreach dataset d_j ∈ Datasets do
        while current run < number of iterations + 1 do
            Set model ← ParseDataReplica(LoadDataReplica(r_i)) ;
            InsertDataset(d_j → model) ;
            Set t_0 ← System.getCurrentTimeMillis() ;
            RemoveDataset(model \ d_j) ;
            Set t_1 ← System.getCurrentTimeMillis() ;
            AssertTrue((model.size()_t0 == model.size()_t1) AND (model.size() == r_i) ) ;
            WriteResultsToLogfile(t_1 − t_0, model.countTriples(), d_j) ;
            RemoveReplicaFromMainMemory(r_i) ;
            CleanUp → InitiateGarbageCollector() ;
        end
    end
end
RemoveReplicasFromInternalMemory(⟨r_1, ..., r_m⟩) ;
```

The benchmark results compartmentalized by framework, device, and data set size for the Androjena and μJena framework are depicted in Figure 6.18 and 6.19 respectively. The plotted graphs represent the removal performance measured in processed triples per second ratios for removing a particular number of triples from RDF data replicas of varying sizes. The size of the data replicas is marked on the x-axis; the processed triples per second ratios are displayed logarithmically along the y-axis. Since the Mobile RDF API does not provide any removal implementations for RDF graphs in its current version, the removal benchmark has only be conducted for the Androjena and μJena frameworks.

[23] For readability issues we omitted initialization as well as logging operations.

6.3.4.1 Androjena

The removal benchmark using the Androjena framework had been conducted with data replicas containing 10,000 triples on the HTC G1 and 20,000 triples on the other devices. In contrast to the insertion benchmark, removal operations with larger replicas produced failures during some test runs or resulted in "out of memory" exceptions wherefore we include and discuss the results that had been acquired for data replicas containing a maximum of 20,000 triples.

Removal times for 1 triple range between 1.0 and 1.3 milliseconds on the HTC G1 (see Table A.21), and 0.1 to 0.9 milliseconds on the other, more powerful devices over all data replica sizes (see Table A.22, A.23, and A.24). Hence, the runtimes of the removal operation in general remain relatively stable over all data replicas for a particular data set size – even on very large data replicas. Just as visible on the insertion benchmark (cf. Section 6.3.3), removal operations of the Androjena framework are not dependent on or influenced by the size of a data replica and are executed with nearly constant processed triples per second ratios. With an increased data set size of 10, 100, and 500 triples, we can observe that removal operations also increase linearly on all devices and for all data replica sizes. As a consequence, the amount of triples to be removed from a data replica scales almost linearly with the total amount of time needed to complete the removal operation.

However, in the range of the analyzed data set sizes, a slight but continuous decrease of processed triples per second ratios with growing data replica sizes was observable on all devices: processing performance constantly drops when the size of the data sets to be removed from a data replica grows. For instance, on the HTC G1, the average processed triples per second ratio drops from 859.91 triples/sec. for removing 1 triple to 573.57 triples/sec. for removing 1,000 triples; similarly, the average removal performance drops from 7,878.79 triples/sec. to 5,368.03 triples/sec. on the Samsung Galaxy S I9000 for identical data set sizes (see Table A.21 and A.23).

The average processed triples per second ratios across all data sets and data replicas account to 716.22 triples/sec. on the HTC G1, 2,457.82 triples/sec. on the Motorola Milestone, 6,935.45 triples/sec. on the Samsung Galaxy S I9000, and 2,041.81 triples/sec. on the Dell Streak. This numbers indicate that the Samsung Galaxy S I9000 and the Dell Streak—although both situated at the upper-level market segment—differ by the factor 3.40 in removal performance. Even the cumulated average removal performance ratios of the Motorola Milestone as a middle-segment device were slightly higher than that measured on the Dell Streak.

Looking specifically at the HTC G1 and the Motorola Milestone, we can observe that an increase of CPU speed from 350 MHz to 600 MHz (factor 1.72) resulted in a multiplicity of processed triples per second ratio of the factor 3.43 using the Androjena framework. From the Motorola Milestone to the Samsung Galaxy S I9000 (CPU clock speed factor 1.67), the removal performance multiplies by 2.82 on average. Comparing the Samsung Galaxy to the HTC G1, a multiplicity by the factor 9.68 in removal performance was observable.

However, on the Samsung Galaxy S I9000 we could observe that Androjena yields the best processed triples per second performance when removing small data sets containing only a few ten to hundred triples. On the HTC G1 as low-segment device, the highest triples per second performance can be observed when removing very small models of 1 or 10 triples, whereas on the middle- and two upper-segment devices, the highest triple per second performance was observable

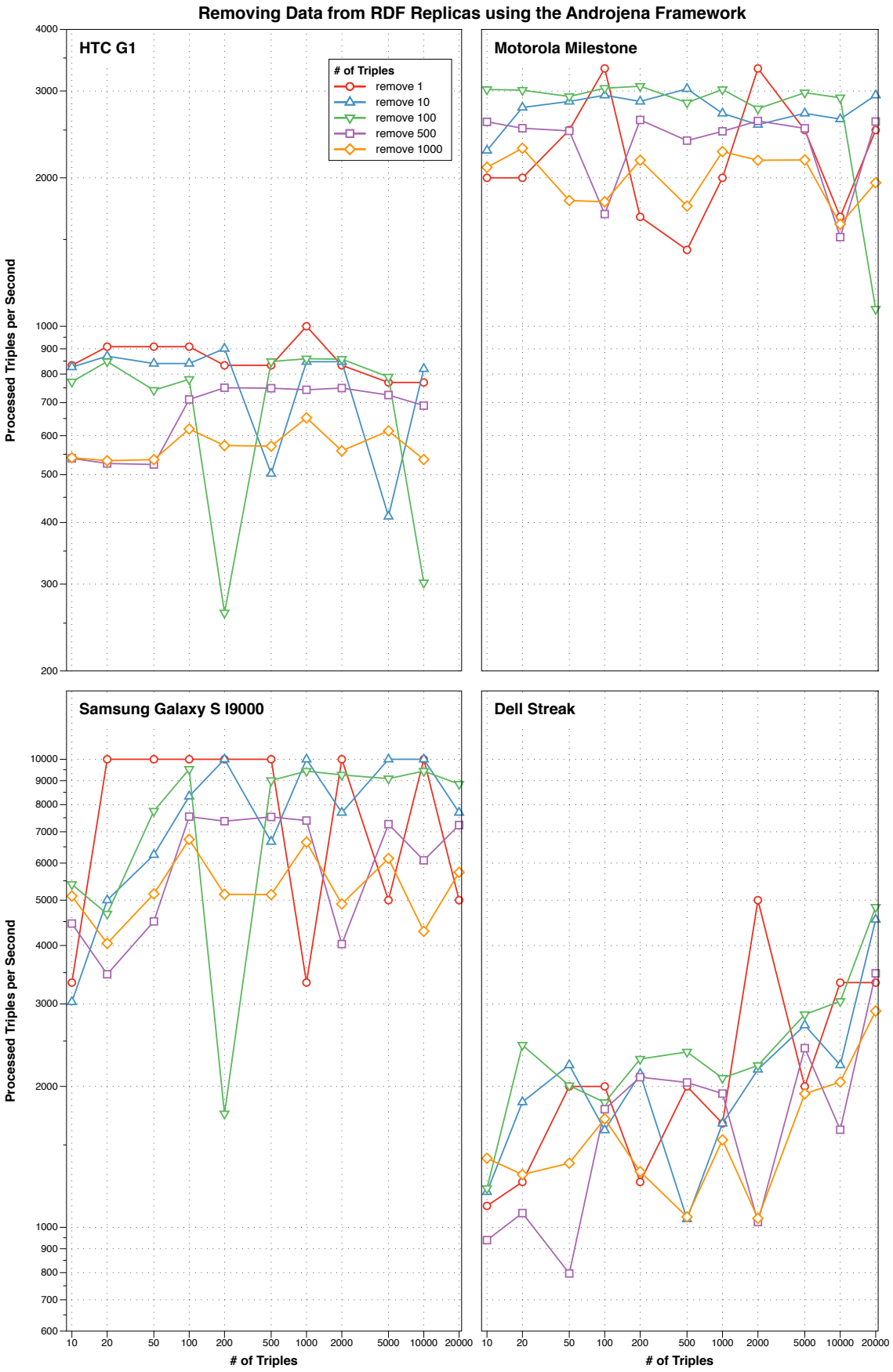


FIGURE 6.18: Performance of removal operations using the Androjena framework

when data sets of 100 triples were removed (see Figure 6.18). For instance, removing 100 triples yields a processed triples per second ratio of 7,652.63 in average on all data replicas[24].

A decrease in triples per second performance when deleting larger data sets containing 1,000 triples from large data replicas containing 20,000 or 50,000 triples—as it was the case on the insertion benchmarks—was not observable during the removal benchmarks for the Androjena framework.

In total, removal operations using the Androjena framework yield best average performance results on the Samsung Galaxy S I9000 where the average processed triples per second ratios across all data replicas and removed data sets in total are substantially higher than on the Motorola Milestone and the Dell Streak. The increased variance on upper-level devices, which was visible during the insertion benchmark for the Androjena framework was not observable in the removal benchmark.

6.3.4.2 μJena

Just as visible in the insertion benchmarks, the processing performance of the μJena framework in particular for larger data replicas dramatically drops and this was likewise the case for removal operations. Furthermore, we were not able to process data replicas with more than 2,000 triples on any of the devices due to the reasons outlined in Section 6.3.3. Therefore, we had to limit data replica sizes to 2,000 on the HTC G1, the Samsung Galaxy S I9000, and the Dell Streak. Despite the larger main memory of the Motorola Milestone compared to the HTC G1, it was not possible to scale our tests up to replica sizes of 2,000 triples and above; as a consequence, benchmarks had only be performed with a maximum of 1,000 triples.

On the HTC G1 as well as on all other devices, removal times grow disproportional along two dimensions: either by increasing the data replicas' sizes or by increasing the amounts of triples to be removed, a rapid growth in total amount of time needed to finish the removal operation was identifiable (see Figure 6.19). For instance, removing 1 triple from a data replica containing 10 triples on an HTC G1 yields an execution time of 2.4 milliseconds on average; removing 1 triple from a data replica containing 2,000 triples requires 188.4 milliseconds on average to finish (cf. Table A.25). This phenomenon was observable on all devices (e.g., on the Dell Streak, removal times grow from 0.4 milliseconds for removing 1 triple from a data replica containing 10 triples to 26.1 milliseconds for removing the same triple from a data replica containing 2,000 triples).

In this respect, processed triples per second ratios dramatically decrease when data sets with hundred or more triples are removed: processed triples per second ratios drop from 119.99 triples/sec. for removing 10 triples to 26.96 triples/sec. for removing 100 triples on average on the HTC G1; on the Motorola Milestone, removal performance ratios drop from 378.80 triples/sec. to 75.79 triples/sec. (see Table A.26); for the Samsung Galaxy S I9000 and the Dell Streak these ratios decrease from 1,212.37 triples/sec. to 339.93 triples/sec. and from 1,495.77 triples/sec. to 377.25 triples/sec. respectively (cf. Table A.27 and A.28). A similar performance drop was observable for removals of 500 triples where the average processed triples per second performance declines approximately by the factor 4 and above on all four devices (see Figure 6.19). However, if the number of triples to be inserted doubles from 500 to 1,000, removal times increase by the factor 2 in average on all devices.

[24] A slightly higher value was measured for removals of 1 triple, but due to limited precision of Java-based measurement methods for micro benchmarks (cf. [Goe04, Goe05]) we leave the average processing performance acquired on the Samsung Galaxy S I9000 for removing 1 triple unconsidered.

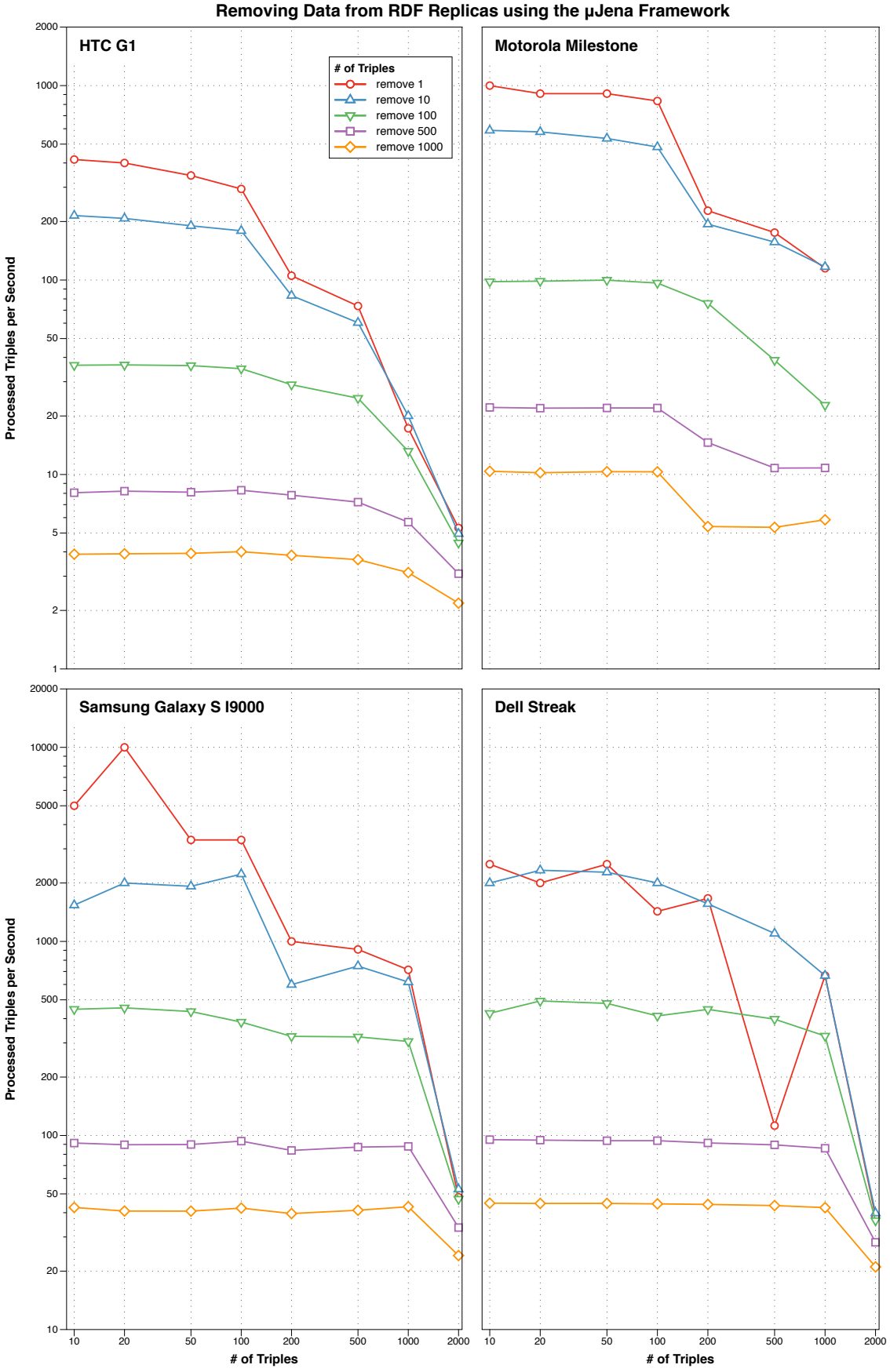


FIGURE 6.19: Performance of removal operations using the μJena framework

The average processed triples per second ratios across all data sets and data replicas account to 72.94 triples/sec. on the HTC G1, 215.24 triples/sec. on the Motorola Milestone, 943.18 triples/sec. on the Samsung Galaxy S I900, and 672.48 triples/sec. on the Dell Streak. These numbers indicate that the Samsung Galaxy S I9000 and the Dell Streak—although both situated at the upper-level market segment—differ by the factor 1.40 in removal performance using the μJena framework. Looking specifically at the HTC G1 and the Motorola Milestone, we can observe that an increase in CPU speed from 350 MHz to 600 MHz (factor 1.72) resulted in an multiplicity of processed triples per second ratio of the factor 2.95 using the μJena framework. From the Motorola Milestone to the Samsung Galaxy S I9000 (CPU clock speed factor 1.67), the average removal performance across all data replicas multiplies by 4.38. Comparing the Samsung Galaxy to the HTC G1, a multiplicity by the factor 12.93 in removal performance was observable. Although μJena yields the highest total removal performance on average over all data replicas and data sets on the Samsung Galaxy S I9000, the processed triples per second ratios on the Dell Streak are slightly higher for the removal of data sets containing 10, 100, 500, and 1,000 triples (see Table A.28)

In general, the variance in processed triples per second ratios was higher on the upper-level devices whereas the Motorola Milestone showed the slightest decrease in removal performance of all devices (see Figure 6.19). In summary, small amounts of triples can be removed from RDF data replicas in reasonable short time using the μJena framework, although the total amount of time needed to execute a removal operation increases disproportional with growing data replica sizes. Albeit processing times for removing triples from large data sets are balanced and nearly constant, the time needed to perform the removal operation exceeds the tolerable maximum from a usability perspective and provide results that are unacceptable for real-world replication scenarios.

6.3.4.3 Mobile RDF

The performance of removal operations could not be measured on the Mobile RDF framework since removal or deletion operations are not implemented in the latest version of the Mobile RDF API[25].

6.3.4.4 Summary

Only two out of three RDF frameworks offer dedicated removal operations in their APIs. The removal performance per data set using the Androjena framework remains relatively stable across all data replicas and scales relatively linear with the number of triples being removed. Using the μJena framework, removal performance significantly decreases for larger data sets and larger data replicas likewise, which renders μJena inappropriate for extensive removal operations. Although Androjena yields highest average removal performance on all devices, it also benefits considerably from available processor power. In total, removal of a few hundred triples can be performed in a few milliseconds and without noticeable delay on all devices using the Androjena framework, whereas for the removal of larger data sets containing 1,000 triples or above, an upper-level device is necessary to process such operations within the range of milliseconds. The slow performance of the μJena framework renders its deployment for real-world replication scenarios practically ineligible.

[25]For this evaluation, we used version 0.3 of Mobile RDF (see Section 3.3.2).

6.3.5 Retrieving Elements from RDF Data Replicas

In the retrieval benchmark, we ascertained the runtime performance of the internal query operations implemented in the frameworks' APIs. For each combination of RDF framework and device, we therefore measured the total amount of time needed to locate and retrieve a distinct triple in RDF data replicas of varying sizes, which have been parsed and transformed into workable in-memory RDF models before a query operation was executed. Since all location and retrieval operations are applied to in-memory RDF models and we analyzed only the execution times of those operations, the different serialization formats are irrelevant wherefore benchmarks are conducted with data replicas serialized in one specific format only.

Algorithm 15: Retrieving a specific triple from an in-memory RDF graph

Data: RDF Resource: *subject*, *predicate*, *object*
Result: Query result set: *result*

Set *Replicas* ← CopyDataChunksToInternalMemory($\langle r_1, \ldots, r_m \rangle$) ;
foreach *replica* $r_i \in$ *Replicas* **do**
  Set *triple* ← CreateTriple(*subject*, *predicate*, *object*) ;
  while *current run* < *number of iterations + 1* **do**
    Set *model* ← ParseDataReplica(LoadDataReplica(r_i)) ;
    RemoveOneTripleFromDataReplica(*model*) ;
    AddTripleToDataReplica(*triple* → *model*) ;
    Set t_0 ← `System.getCurrentTimeMillis()` ;
    Set *result* ← InitiateQuery(*model*, *triple*) ;
    AssertTrue(*triple* ∈ *result* AND $|result| = 1$) ;
    Set t_1 ← `System.getCurrentTimeMillis()` ;
    WriteResultsToLogfile($t_1 - t_0$, *model.countTriples()*, ...) ;
    RemoveReplicaFromMainMemory(r_i) ;
    CleanUp → InitiateGarbageCollector() ;
  end
end
RemoveDataChunksFromInternalMemory($\langle r_1, \ldots, r_m \rangle$) ;

Algorithm 15 provides a conceptual overview of the main steps performed during the retrieval benchmark. For each iteration of the benchmark and data replica, we first removed an arbitrary triple from the data replica and inserted a specific triple that is used for the query operation. Before the runtime as well as other relevant information related to a query operation are recorded in the corresponding log file, a validation routine checks that the result set evaluates to true iff it contains only the specified triple, otherwise an error is reported.

Since query operations are implemented (and designated) differently across frameworks, we specifically considered only those query operations for the retrieval benchmark that expose a similar method signature (the parameters a method takes) and deliver similar, i.e, comparable result types[26]. Usually, triples that match the specified query parameters are returned in form of a list-based data structure that can be traversed by an iterator. In addition, we did not consider combined queries as these are not supported natively by any of the query methods implemented in the frameworks' APIs. Figure 6.20 and 6.21 provide three examples of the query methods' signatures implemented in each framework as well as the data structures used by the query methods to store matching triples.

[26]However, we do not distinguish whether a query operation returns the matching triples in form of a collection or as a new RDF graph.

```
Androjena:
==========
   Model result = ModelFactory.createDefaultModel();
   Resource subject = model.createResource(
      "http://dbpedia.org/resource/Abraham_Lincoln");
   Property predicate = model.createProperty(
      "http://dbpedia.org/ontology/birthPlace");
   Resource object = model.createResource(
      "http://dbpedia.org/resource/Kentucky");

   SimpleSelector selector = new SimpleSelector(subject, predicate, object);
   result = model.query(selector);

Microjena:
==========
   StmtIterator result = model.listStatements(subject, predicate, object);

   while( (Statement statement = result.nextStatement()) != null) ){
      //iterate through results
   }
```

FIGURE 6.20: Example of the query method signatures and result sets implemented in the Androjena and μJena frameworks

```
Mobile RDF:
===========
   Collection<Statement> matching_statements = new ArrayList<Statement>();

   RDFGraph graph.queryStatement(
      "http://dbpedia.org/resource/Abraham_Lincoln",
      "http://dbpedia.org/ontology/birthPlace",
      "http://dbpedia.org/resource/Kentucky",
      matching_statements);
```

FIGURE 6.21: Example of the query method signature and result set implemented in the Mobile RDF API

The query methods of Androjena and μJena exhibit identical method signatures, i.e., they both take three parameters referring to the corresponding RDF elements as query attributes. For readability reasons, we included the declarations of subject, predicate, and object only on the Androjena excerpt and omitted it for μJena as they are identical. While Androjena stores the matching triples in a new `model` instance, μJena returns them wrapped in a statement iterator (`StmtIterator`) instance.

In contrast to the other methods, the query method of the Mobile RDF API does not implement a dedicated type system for RDF elements; instead it takes the elements to be searched for as plain strings (`java.lang.String`) and stores the matching triples in form of statements (`de.hedenus.rdf.BasicStatement`) in a `Collection`-based data structure passed as fourth parameter to the query-method.

The results of the retrieval benchmark compartmentalized by framework and device are depicted in Figure 6.22, 6.23, and 6.24. The plotted graphs represent the total amount of time needed to find a particular triple within data replicas of specific sizes, marked on the x-axis. The y-axis shows the total amount of time in milliseconds the query method takes to find and retrieve a particular triple depending on the size of a data replica.

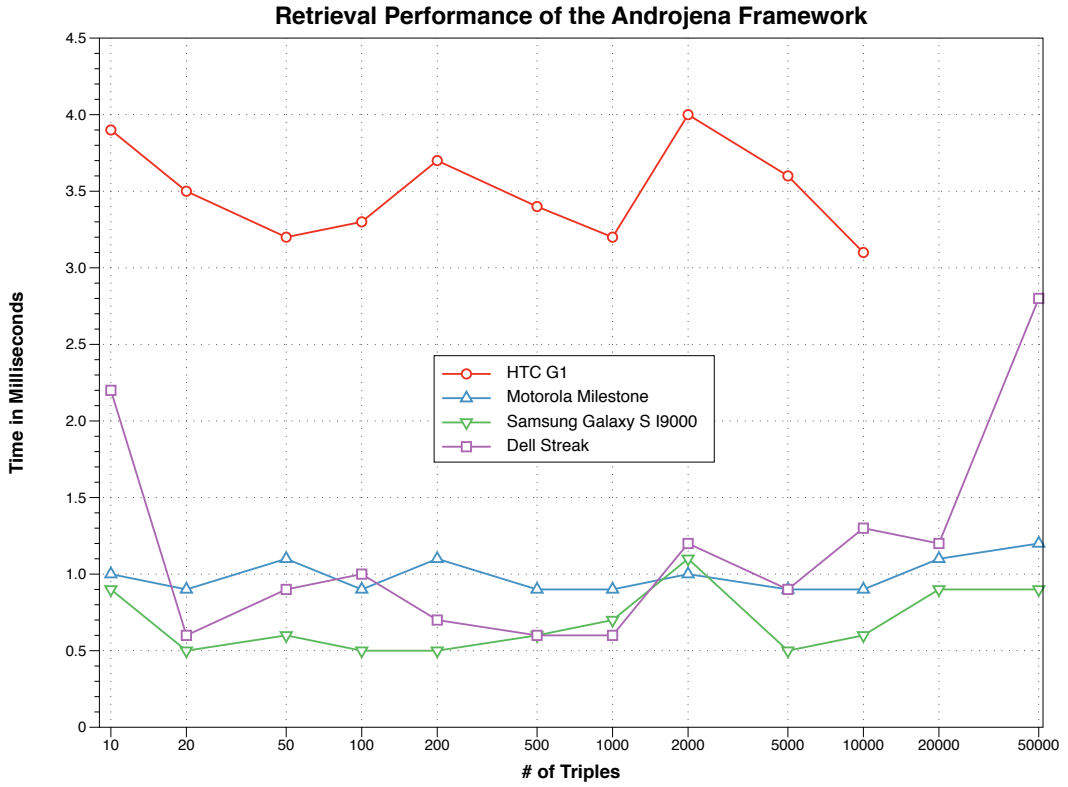


FIGURE 6.22: Performance of retrieval operations using the Androjena framework

6.3.5.1 Androjena

On the Motorola Milestone, the Samsung Galaxy S I9000, and the Dell Streak, query operations had been analyzed for data replicas containing a maximum of 50,000 triples; on the HTC G1, we had to limit data replica sizes to 10,000 triples due to limited main memory capacity (cf. Table 6.1).

The execution time of query operations remains relatively stable over all data replicas and on all devices—also for very large data replicas containing several thousand triples—and query operations are processed with nearly constant runtimes. As a result, we could observe that the execution time of query operations is independent from the size of data replicas, i.e., there is no linear dependency between the amount of triples held in an in-memory RDF model and the execution time of query operations applied to this in-memory graph using the Androjena framework (see Figure 6.22). Query times for 1 triple range between 3.1 and 3.9 milliseconds on the HTC G1, 0.5 to 1.2 milliseconds on the Motorola Milestone and the Samsung Galaxy S I9000, and 0.6 to 2.8 milliseconds on the Dell Streak (see Table A.29).

The average query times across all data replicas for locating and retrieving one specific triple account to 3.49 milliseconds on the HTC G1, 0.99 milliseconds on the Motorola Milestone, 0.69 milliseconds on the Samsung Galaxy S I9000, and 1.17 milliseconds on the Dell Streak. These numbers indicate that the Samsung Galaxy S I9000 and the Dell Streak—although both situated at the upper-level market segment—differ by the factor 1.68 in query performance. Even the cumulated query performance of the Motorola Milestone as a middle-segment device is slightly faster than that of the Dell Streak (factor 1.17). This is remarkable for a device situated in the upper-level segment since its execution times are approximately 20% slower compared to the execution times of a device located at the middle-market segment. We could not find any

logical reason for this behavior although we believe that for such elementary operations, which are executed in the range of very low milliseconds, there is a higher variance in execution times on more powerful devices probably attributed to an optimized *modus operandi* of the internal thread scheduler. This behavior is likely to normalize for expanded search queries that require more CPU cycles to complete. On the Dell Streak, however, we could also notice an increased variance among the query execution times, which was not that distinctive on the other devices. However, comparing the HTC G1 to the Motorola Milestone and the Samsung Galaxy S I9000, query operations finish by the factor 2.93 faster on the Motorola Milestone compared to the HTC G1 and by the factor 4.20 faster on the Samsung Galaxy S I9000.

In total, the best average query performance results have been measured on the Samsung Galaxy S I9000 where the average execution times over all data replicas are approximately 30% lower than on the Motorola Milestone (factor 1.43). In summary, query operations over all data replicas can be executed within the range of low milliseconds on all devices using the Androjena framework, irrespectively of concrete data replica sizes.

6.3.5.2 μJena

Using the μJena framework, it was not possible to query data replicas with 10,000 triples or above on any device; therefore we scaled query performance tests up to 1,000 triples on the Motorola Milestone, 2,000 triples on the HTC G1, and 5,000 triples on the two upper-level devices. As visible in the other benchmarks, the μJena framework was not capable of processing RDF data replicas with more than 1,000 triples on the Motorola Milestone despite its larger main memory compared to the HTC G1 (see Table 6.1); in addition, no memory extensive processes or applications were running in parallel.

Unlike the Androjena framework, execution times of query operations slightly increase with growing data replicas; on the Dell Streak, for instance, average query times grow from 19.1 milliseconds for querying a replica with 1,000 triples to 46.7 milliseconds for querying a replica containing 2,000 triples; on the HTC G1, query times grow from 203.40 milliseconds to 391.20 milliseconds for identical replica sizes. In total, query times grow by factor 2.62 between the smallest data replica containing 10 triples and the largest data replica containing 2,000 triples on the HTC G1; on the other devices, query times grow by factor 1.21 on the Motorola Milestone, by factor 1.63 on the Samsung Galaxy S I9000, and by factor 3.23 on the Dell Streak (see Table A.29).

As a result and unlike to the Androjena framework, the execution time of query operations is not independent from the size of data replicas (see Figure 6.23). In consequence, there exists a perceptible dependency between the amount of triples contained in an in-memory representation of an RDF data replica and the execution times of query operations on this in-memory model using the μJena framework.

Query times for 1 triple depending on the data replica size range between 149.6 and 391.2 milliseconds on the HTC G1, 267.1 and 5,125.1 milliseconds on the Motorola Milestone[27], 12.3 and 21.9 milliseconds on the Samsung Galaxy S I9000, and 16.40 and 53.0 milliseconds on the Dell Streak (cf. Table A.29). The average query times across all data replicas for locating and retrieving one specific triple account to 202.23 milliseconds on the HTC G1, 1,473.56 milliseconds on the Motorola Milestone, 14.15 milliseconds on the Samsung Galaxy S I9000, and 24.60 milliseconds on the Dell Streak. These numbers indicate that the μJena framework performs query

[27] The unusually high upper-bound value measured on the Motorola Milestone are rather untypical for middle-market segment devices in particular in relation to the results acquired on the other devices.

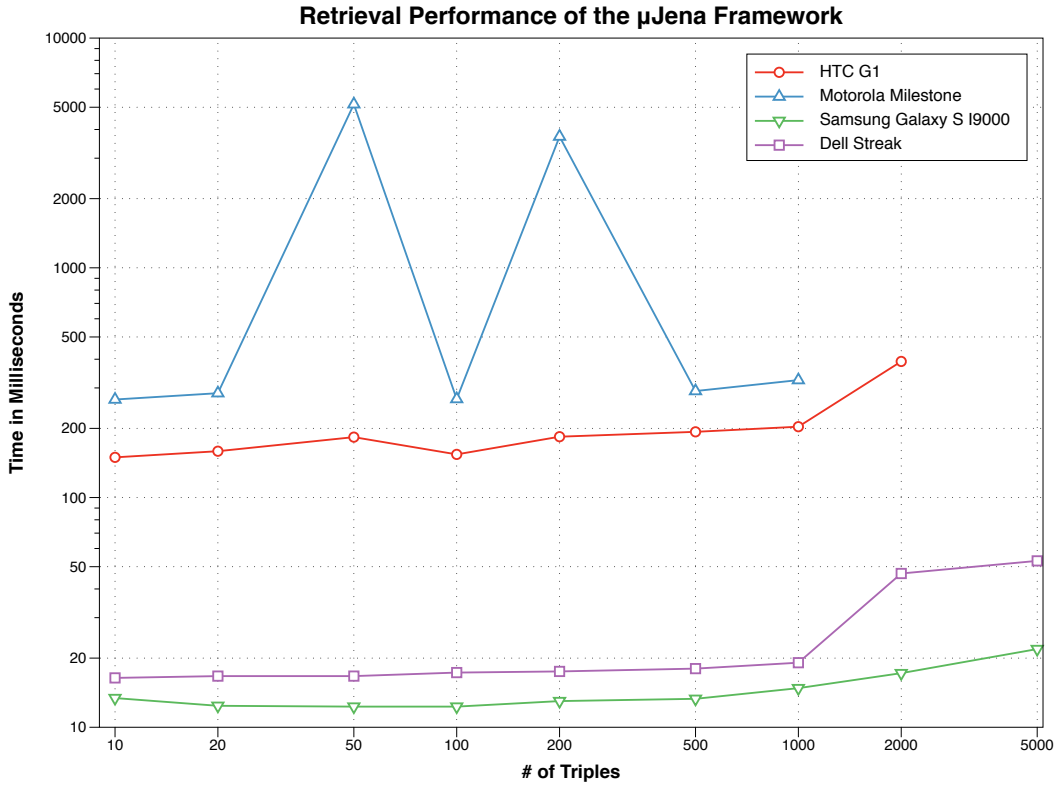


FIGURE 6.23: Performance of retrieval operations using the μJena framework

operations almost 1.70 faster in average on the Samsung Galaxy S I9000 compared to the Dell Streak—although both devices are situated at the upper-level market segment.

On the Dell Streak again, we could notice an increased variance among query execution times, which was not that distinctive on the two other devices[28]. However, comparing the HTC G1 to the Samsung Galaxy S I9000 and the Dell Streak, query operations finish by the factor 13.94 faster on the Samsung Galaxy compared to the HTC G1 and by the factor 8.22 faster on the Dell Streak.

In total, the best results in terms of query performance were measured on the Samsung Galaxy S I9000 where the average execution times over all data replicas are approximately 41.2% lower than on the Dell Streak. In summary, query operations on data replicas using the μJena framework can be executed within the range of low milliseconds only on devices from the upper-level segment.

6.3.5.3 Mobile RDF

The size of data replicas on the Mobile RDF framework could be scaled up to 10,000 triples on the HTC G1, and 20,000 triples on the other devices; although the two upper-segment devices incorporate 512 MB of main memory compared to the Motorola Milestone, which offers 256 MB of main memory, data replicas with more than 20,000 triples could not be processed despite their larger physical main memory.

[28] We excluded the results obtained on the Motorola Milestone from this calculation since its execution times for data replicas containing 50 and 200 triples in particular show untypically high values that could not be reproduced in subsequent benchmarks.

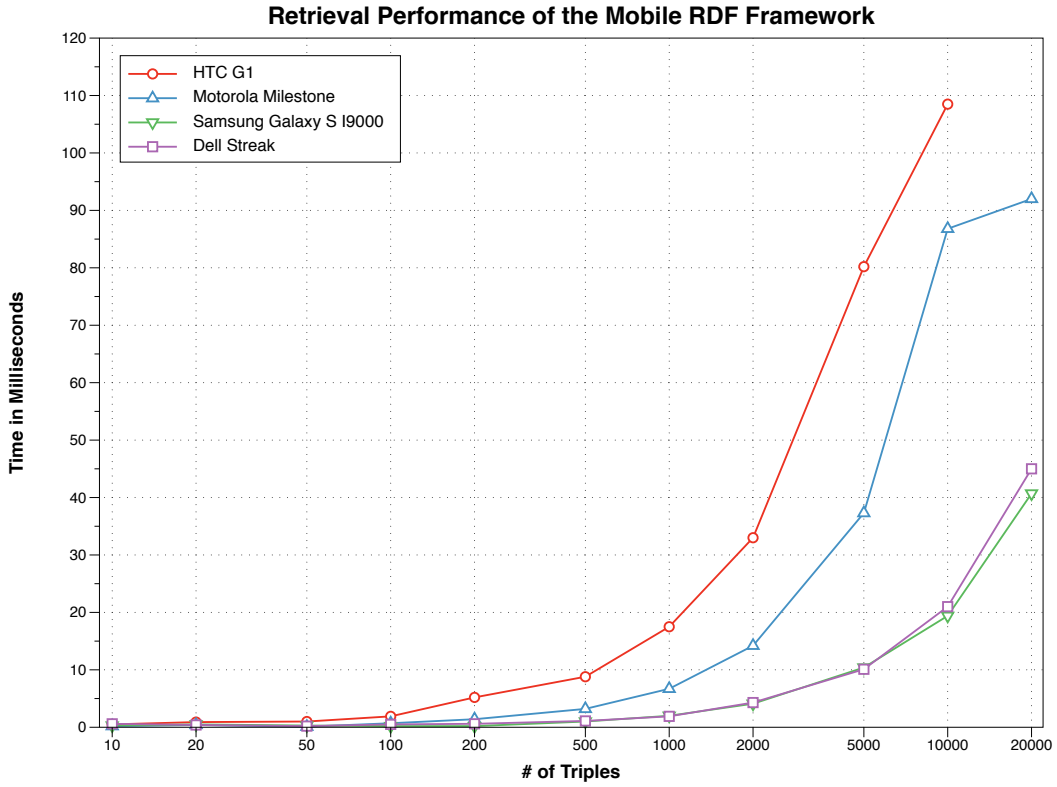


FIGURE 6.24: Performance of retrieval operations using the Mobile RDF framework

Just as visible on the μJena framework, the execution times of query operations using the Mobile RDF framework highly correlate with the size of data replicas; we could notice a dramatic increase in query execution times when the size of data replicas grows (see Figure 6.24). For instance, on the Motorola Milestone, average query times grow from 0.2 milliseconds for querying a replica with 10 triples to 92.0 milliseconds for querying a replica containing 20,000 triples; on the Samsung Galaxy S I9000, query times grow from 0.3 milliseconds to 40.7 milliseconds for identical replica sizes (see Table A.29).

Considering the total growth of the average execution times from the smallest to the largest data replica that Mobile RDF is capable to process on a device, we could notice an increase in query execution times by the factor 217.0 on the HTC G1. On the other devices, query execution times grow by the factor 460.0 on the Motorola Milestone, 135.67 on the Samsung Galaxy S I9000, and 75.0 on the Dell Streak. More specifically, query execution times increase from 0.5 milliseconds to 108.5 milliseconds on the HTC G1, 0.2 milliseconds to 92.0 milliseconds on the Motorola Milestone, 0.3 milliseconds to 40.7 milliseconds on the Samsung Galaxy S I9000, and 0.6 milliseconds to 45.0 milliseconds on the Dell Streak (see Table A.29).

The average query times across all data replicas for locating and retrieving one specific triple account to 25.75 milliseconds on the HTC G1, 22.09 milliseconds on the Motorola Milestone, 7.19 milliseconds on the Samsung Galaxy S I9000, and 7.79 milliseconds on the Dell Streak. These numbers indicate that the Mobile RDF framework performs query operations most rapidly on the Samsung Galaxy S I9000; the average query performance across all data replicas was approximately 1.08 times faster than on the Dell Streak. Comparing the query performance of Mobile RDF on the Samsung Galaxy S I9000 to the HTC G1 and the Motorola Milestone, query operations finish by the factor 3.58 faster on the Samsung Galaxy compared to the HTC G1 and by the factor 3.07 faster compared to the Motorola Milestone.

As a consequence and unlike to the Androjena framework, the execution time of query operations is also not independent from the size of data replicas using the Mobile RDF framework, where an almost exponentially growing dependency between the amount of triples contained in an in-memory model and the execution time of query operations on those in-memory models was observable. However, on the two less powerful devices, the growth in execution times was more distinctive than on the two upper-level devices (see Figure 6.24). Furthermore, we could identify a greater variance in query times on the two less powerful devices, which was not that distinctive on the two more powerful devices. In this respect, average query execution times grow stronger for larger data replicas on less powerful devices.

In total, the best average query performance results have been measured on the Samsung Galaxy S I9000. In summary, query operations on data replicas using the Mobile RDF framework can be executed within the range of milliseconds on all devices, where upper-segment devices are approximately 3.5 times faster than low- and middle-segment devices. In general, query operations can be executed in reasonable time on all devices, although query times increases almost exponentially with growing data replica sizes.

6.3.5.4 Summary

In summary, the best retrieval performance was measurable for the Androjena framework where queries are executed in the range of low single-digit milliseconds over all data replicas. Although the retrieval performance of μJena and Mobile RDF lies in the range of low milliseconds too, their query runtimes start to increase when larger data replicas are being processed. The highest growth was measurable for the Mobile RDF framework where query times increase almost exponentially, particularly on devices from the entry- and middle-market segment. In consequence, only Androjena allows to query data replicas in constant time depending on the size of the result set. In general, all frameworks benefit from additional processing power and manage to complete simple retrieval operations in acceptable time.

6.3.6 Constructing In-memory RDF Graphs

Although context-relevant information is usually not created but replicated to the mobile device from external sources via wireless network connections, we additionally analyzed the performance of constructing in-memory RDF models as such an analysis allows us to obtain insights related to the general RDF processing performance of a particular combination of framework and mobile device[29]. We therefore measured the total amount of time needed to create workable in-memory RDF graphs where we have scaled the size of those RDF graphs towards the limits imposed by the physical main memory installed in a device, i.e., the maximum amount of triples that can be processed without provoking "out of memory" exceptions. Those numbers are relevant for tasks related to the creation of RDF-based context models and their aggregation into a global context configuration that serves as a basis for further internal processing such as reasoning and consolidation. In this benchmark, we were able to create in-memory RDF models containing a maximum of 50,000 triples on the Samsung Galaxy S I9000 and 20,000 triples on the two less powerful devices. Figure 6.25 depicts the results of the construction benchmarks using the three mobile RDF frameworks Androjena, μJena, and Mobile RDF. Detailed results can be found in Table A.30 in Appendix A for each analyzed device and framework.

[29]This benchmark has only been conducted on the HTC G1, Motorola Milestone, and the Samsung Galaxy S I9000.

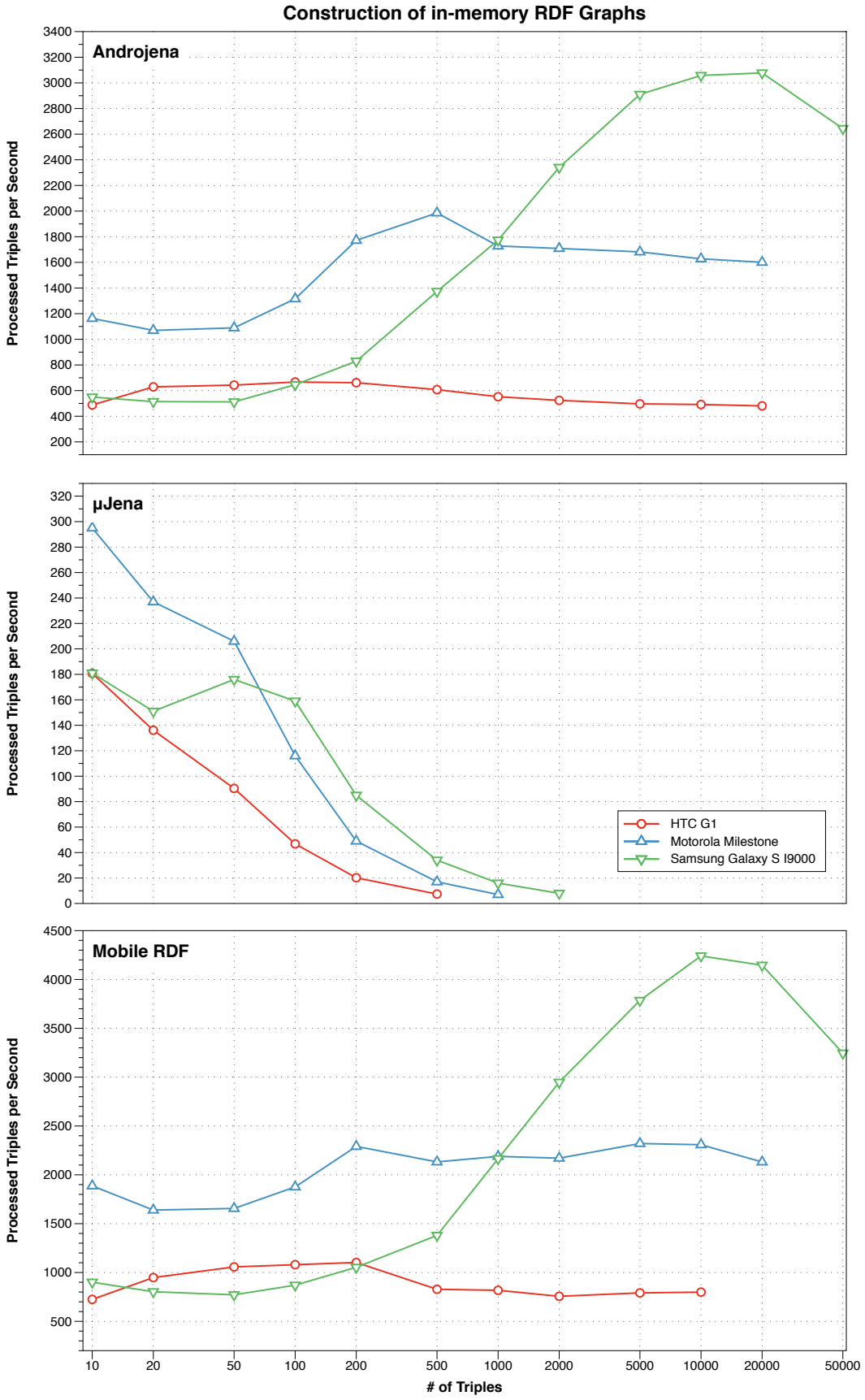


FIGURE 6.25: Construction of in-memory RDF graphs using the Androjena, μJena, and Mobile RDF frameworks

When creating in-memory RDF graphs of certain sizes, we can observe a similar behavior on all three tested platforms. Androjena and Mobile RDF exhibit very similar results, namely, a nearly constant processing time per triple, even with increasing model size. Although processing times of mobile RDF frameworks vary considerably across small context descriptions with sizes smaller than 500 triples (up to factor 10 on the Samsung Galaxy S I9000 using Mobile RDF for processing a model containing 100 triples), processing times normalize for models of size greater or equal than 1000 triples on the two frameworks. In general, we can observe that Androjena and Mobile RDF are able to create and handle RDF graphs containing 20,000 or more triples, although the limiting factor is the device's internal memory capacity.

Additionally, the total execution time (in milliseconds) for Androjena and Mobile RDF scales almost linearly with the size of context descriptions. The performance of μJena, on the contrary, decreases significantly with increasing model sizes, leading to very low processing times with models larger than 100 triples. μJena tests with more than 2,000 triples failed on all devices, making it basically unsuitable for the processing of voluminous RDF data.

Processing speed of Androjena ranges between 480 and 680 triples per second on an Android HTC G1, and 1000 and 2000 triples per second on a Motorola Milestone. Interestingly, on the Samsung Galaxy we can observe that the performance increases when models with more than 200 triples are created. The performance of μJena decreases with increasing model size on all three devices. Mobile RDF exhibits a similar performance behavior compared to Androjena where a significant increase in triples per second values on a Samsung Galaxy can be observed for models with more than 500 triples. In general, Mobile RDF has shown to be the most performant framework w.r.t. the amount of triples processed per second on all tested devices.

When comparing the different devices, we can observe the expected behavior that the Android HTC G1 exposes the weakest results due to its slow CPU and small main memory, leading to memory problems when creating models with 20,000 or more triples. The other devices expose a better performance, making them more suitable for creating and processing larger volumes of RDF data. Only the Samsung Galaxy I S9000 was able to handle a model of 50,000 triples; on the other devices tests with this model size failed with "out of memory" errors.

6.4 Discussion and Summary

In this evaluation, we specifically analyzed the runtime behavior of typical RDF operations involved in data replication tasks to obtain insights about the efficiency of processing data replicas of varying sizes on devices from different mobile market segments using currently available mobile RDF frameworks. In summary our analysis revealed that modern mobile devices in combination with recent RDF frameworks optimized for mobile platforms can without hesitation be used as the basis for replicating and processing RDF documents directly on a device using the presented context-sensitive RDF data replication framework without the necessity to rely on a server infrastructure. Although the behavior of the deployed RDF frameworks differs across machines, we can observe certain trends regarding the applicability of current RDF frameworks for specific replication-related purposes. In particular, our analysis revealed that the presented replication framework is principally capable of processing large data replicas containing several thousand triples in sufficient time using the Androjena or the MobileRDF framework, where the maximum size of data replicas is in most cases limited by the amount of physical main memory installed on a device rather than by insufficient processing power.

Taking a closer look at the deployed RDF frameworks in particular, Mobile RDF revealed to be the fastest framework with highest average parsing performance across all data replicas and devices closely followed by Androjena. The parsing performance of the replication framework dramatically drops when the μJena framework is being used – in particular when large data replicas are being processed where an almost exponential decrease was observable. Although μJena exhibits by far the weakest parsing performance, it outperforms the other frameworks in terms of storage and serialization performance. In general, the storage performance scales relatively linear with the amount of triples to be stored on all three frameworks. Additionally, a comparison of the parsing and storage performance reveals interesting results: on the two less powerful devices, serialization performance exceeds parsing performance; on the two more powerful devices, in contrast, parsing performance lies slightly above serialization performance where the limiting factor remains to be the read-/write-performance rather than the CPU performance, which was the case on the two technically weaker devices. When data are to be added to data replicas, best results have been measured using the Mobile RDF framework followed by Androjena; both frameworks allow to perform insert operations in constant time independent from specific data replica sizes. Only two of three frameworks offer support for removal operations where only with the Androjena framework, we were capable of performing removal operations in almost linear time depending on the amount of triples being removed. The weak performance using the μJena framework, which was visible in the parsing benchmark, was also present during the insertion, removal, and creation benchmarks. When traversing an RDF graph in order to retrieve a specific triple, lowest retrieval times were measured using the Androjena framework, which also remain constant across all data replicas. The retrieval times using the μJena and Mobile RDF framework, in contrast, scaled with data replica sizes and were noticeably higher for larger data replicas.

In summary, we could observe that the performance of the presented replication framework in combination with current RDF frameworks scales relatively well with available processor speed as evident from benchmark results that doubles or, in some cases, threefold between devices from different market segments. In some benchmarks we could observe that an increase in CPU clock speed from 350 MHz to 600 MHz resulted in a twofold up to fourfold increase in processing performance. Although best performance results on four of six benchmark were achieved using the Mobile RDF framework, Androjena appeared to be the most mature RDF framework and performed excellent during all benchmarks on all tested devices. It incorporates the most efficient memory management and allows to process data replicas containing up to 100,000 triples on state-of-the-art devices. Additionally, it scales very well with available CPU power but also provides acceptable results on less powerful devices. Unlike μJena and Mobile RDF, the performance of retrieval and modification operations are not influenced by data replica sizes and render its deployment particularly for processing large data replicas appropriate. Although highest triples per second ratios in the creation, parsing, storage, and insertion benchmarks were achieved using the Mobile RDF framework, its internal memory management lacks efficiency and does not allow to process data replicas containing more than 20,000 triples – even on devices with 512 MB of main memory. The weak performance of the μJena framework, which was visible during four of six benchmarks, renders its deployment in real-world replication scenarios more than questionable. More seriously, μJena incorporates the least efficient memory management that restricts the processing of data replicas to a maximum of 5,000 triples.

Chapter 7

Conclusion and Future Work

"Prediction is very difficult, especially about the future."
Niels Bohr (1885 - 1962)

7.1 Conclusion

In this work, we introduced a novel approach that combines context-aware computing concepts with semantic technologies, distributed transaction management concepts, and mobile information system peculiarities in order to build an infrastructure for intelligently assisting mobile users by selectively replicating RDF data from remote data sources to a mobile device according to their current and future information needs and the different contexts mobile users are operating in. We showed that such a synthesis is a driving force not only in enhancing mobile user information seeking satisfaction and increasing the precision of context-dependent information retrieval processes, but also provides the Semantic Web community with new application fields as well as the Mobile Computing community with information about valuable semi-structured data sources and novel technological opportunities. Since recent research in context-aware computing and mobile information systems shows that context and context awareness should be central parts of future mobile information systems, we introduced an application scenario that outlines how Semantic Web-enhanced context-aware computing can support mobile users in fulfilling their information needs, which also serves as the motivating example of this work. Our work builds on the *design science research methodology* (cf. [MS95, MMG02]), the principle of which is to "extend the boundaries of human and organizational capabilities by creating new and innovative artifacts" where the "knowledge and understanding of a problem domain and its solution are achieved in the building and application of the designed artifact" [HMPR04].

We provided an overview on how context is used in information systems and discussed problems and limitations of current context-aware computing approaches. We presented areas where the introduction of concepts, technologies, and languages from the Semantic Web makes substantial contributions regarding the representation, processing, and management of contextual information. For this purpose, concepts from graph theory, distributed transaction management, and the Semantic Web were adopted in order to demonstrate that a synthesis of those fields can serve as an architectural infrastructure for the efficient acquisition, management, and processing of contextual information directly on a mobile device. In consequence, dependencies to external

systems are reduced while preserving security and privacy concerns since private data do not need to be transferred outside the mobile system.

We elaborated on the notions of context and context awareness, their underlying concepts, how they are used and understood in different domains, and the problems associated with their deployment and utilization in information systems. We assented that context is to be understood as a dynamic and emergent phenomenon that evolves in the process of communication and is continuously renegotiated between communicating partners which makes its determination at design time of a system difficult if not impossible. We proposed the use of Semantic Web technologies and concepts for the representation and processing of contextual information in order to overcome the issues originating from the deployment and utilization of context in information systems leading to an approach that we denote as *Semantic Web-enhanced context-aware computing*.

The application of Semantic Web technologies, languages, and concepts for the representation and processing of contextual information on mobile devices requires the availability of RDF-based management and processing frameworks designed specifically for mobile deployment. We therefore qualitatively analyzed existing RDF/OWL parsers, RDF frameworks, and RDF query and storage infrastructures according to their functionalities in processing RDF graphs and outlined their main aspects in a comparative study. Our study revealed that most of the analyzed frameworks are in a prototypical status but provide the necessary functionalities for processing RDF data on mobile devices although they lack both efficiency and performance when processing larger RDF graphs containing hundreds of thousands triples (see also Chapter 6).

In a second study, we analyzed existing Semantic Web projects that utilize context-aware computing concepts with semantic technologies and languages in order to provide personalized and context-driven services. Those applications are well-known in the Semantic Web community but provide only rudimentary context-aware functionality or were mainly developed for covering a specific application domain. Our analysis also revealed that context-driven RDF data replication to mobile devices has not been addressed by current or related research yet. Moreover, a Semantic Web-based context-sensitive RDF-data replication framework for mobile devices that incorporates the functionalities and concepts introduced in this work does not exist to date.

For delineating our approach, we collected and analyzed requirements of context management and processing architectures from related domains that serve as design considerations of the formal models and the conceptual system architecture. Our approach allows for a controlled acquisition, aggregation, and consolidation of contextual information while taking into account mobile operating system peculiarities and distributed transaction management concepts. At the same time, it guarantees consistency, accurateness, and completeness among contextual data and acquisition workflows. It employs a loose coupling between context acquisition and data provisioning components, which is gained by applying semantic technologies (data models, vocabularies, inference) to interpret and process context information.

Central to our approach is the assumption that data emitted and required by a context provider can sufficiently be described using terms from controlled vocabularies by means of a data description. Therefore, we developed a lightweight data description ontology that allows for the calculation of compatibility scores indicating the degree of compatibility that exists between pairs of context providers and hence serves as a basis for deducing orchestration networks that specify context acquisition workflows. For the processing of context acquisition workflows, we defined an extended transaction-based processing model that builds on the ACTA formal framework and allows for a formal description of the activities being relevant for a controlled and deterministic acquisition, aggregation, consolidation, and dissemination of contextual information. Although

our approach exhibits extensive control and management mechanisms, it guarantees data and process consistency, accurateness, and completeness among contextual information and context acquisition workflows.

In addition to the formal, conceptual, and algorithmic description of our approach and the adhering conceptual system architecture, we presented selected details of the proof-of-concept prototype that relieves 3rd-party developers from the necessity of having to deal with processing and management-specific aspects and instead focus on context acquisition and data replication logic only. Practical insights into implementation details of context and data providers were discussed and concrete examples were presented that demonstrate how context-relevant data can be acquired from locally deployed sensors, represented using semantic vocabularies, and complemented with context-relevant data from external sources.

As a proof of concept, we presented a case study based on the information needs of a typical mobile knowledge worker in which relevant information from Linked Data sources is replicated based on the user's current location and upcoming appointments. This case study was defined on the basis of the most important mobile context type, that is location, and shows that a significant amount of available information that is irrelevant to the user's current context does not need to be considered in data replication tasks. As a result, the approach not only increases precision of mobile information retrieval tasks but it also reduces the need for explicit user attention and the cognitive load associated with information seeking tasks.

Furthermore, we conducted a comprehensive quantitative performance analysis in order to obtain insights regarding the processing efficiency of local RDF data replicas on modern mobile platforms using the proposed context-sensitive RDF data replication framework. We analyzed the runtime behavior of typical RDF processing operations applied to local RDF data replicas to evaluate whether data replicas can be processed on current mobile devices in reasonable time using available RDF frameworks. The evaluation showed that our approach in combination with current RDF frameworks provides a suitable basis for the replication and processing of RDF data consisting of several thousand triples when deployed on state-of-the-art mobile devices. It also indicated that the framework serves as an appropriate platform for the development of context-sensitive applications and services, the decision-making processes of which are oriented on the users' current contexts and their information needs thus forming a new generation of future mobile applications.

7.2 Future Work and Possible Application Fields

Semantic Web technologies, concepts, and languages proved to provide substantial contributions regarding the deployment and representation of context and context awareness in information systems. However, there exists several open issues that need to be addressed in future research:

- *Integration into and combination with Semantic Desktop infrastructures to facilitate personal information management on mobile devices*

  The deployment and integration of Semantic Web technologies into the conceptual models of desktop computers represents a recent research endeavor designated as *Semantic Desktop* [SBD05, SKSB09, FAS09]. In the Semantic Desktop, the information space is defined relative to the user rather than bound to a specific computational device [HMD05]. In the Semantic Desktop, the notion of context is used to annotate information resources with specific contextual dimensions that allow for their classification according to different

aspects an information resource pertains to. Since mobile devices are increasingly used for the management of digital information assets of mobile users, a synthesis of Semantic Desktop concepts with context-aware computing concepts will provide substantial benefits for personal information management as Semantic Desktop graphs can be interlinked with contextual information acquired and processed by a context framework. Contextual information acquired by a context framework can both complement information hosted in Semantic Desktop systems and be used for replicating data from such systems to the mobile device in order to satisfy current and future information needs of mobile users depending on the current context a user is operating in. Moreover, the situative provision of context-relevant information hosted within Semantic Desktop systems allows for the development and deployment of applications that intelligently assist the user in fulfilling their tasks where decision making processes are based on the evaluation and analysis of such data rather then predetermined preferences. In future work, we aim to wrap information sources locally deployed on a mobile device with RDF wrapper components that expose the data hosted in such sources as RDF graphs to enable their interlinkage and utilization with Semantic Desktop systems.

- *Consolidation and classification of reference architectures proposed for context management and processing*

  To date, many architectures have been proposed for the processing and management of contextual information claiming to be reference architectures (e.g. [BDR07, DWM08, SB08b]). We believe that a consolidation and functional classification of such reference architectures contributes towards a wider adoption of context-aware computing concepts. Moreover, it can help in achieving a general model of context and context awareness across different domains and disciplines that is to date de facto non-existent.

- *Context-dependent Linked Data federation and visualization in mobile Augmented Reality interfaces*

  The continuous technical advancements of mobile information technology together with the omnipresence of mobile devices offers novel application scenarios that can profit from a deployment of context-aware computing concepts in general and Semantic-Web enhanced context-aware computing concepts in particular. One such technology is that of Augmented Reality where data being replicated from linked data sources can be federated with data replicated from geographical databases in order to visualize Linked Data in Augmented Reality interfaces on the mobile device. The potential of integrating Linked Data in mobile Augmented Reality applications is also discussed in a recent position paper [RHP+10]. Next to the deployment of a framework comparable to the one presented in this thesis, we can add the dimension of context to Augmented Reality visualization interfaces in which data being relevant to the user's current location are replicated in a proactive manner to the mobile device and are federated with data from related sources corresponding to the user's information needs.

- *Inclusion of feedback loops to make the context framework context-dependent*

  Currently, the framework presented in this work does not include feedback loops that would allow for the adjustment of context acquisition and aggregation tasks according to data provisioning needs, and it lacks advanced reasoning capabilities, which we plan to implement in the near future.

- *Making a context-framework context-aware*

  A context framework developed for resource-constraint devices such as mobile phones should be made context-aware to adapt its processing rules and policies according to specific technical circumstances, for instance to reduce replication cycles or computational expensive tasks in case of low battery capacity etc. Hence, the incorporated business logic is able to react according to changing technical or infrastructural circumstances and to adapt processing tasks accordingly. The framework proposed in this thesis is principally capable of adapting its processing tasks with the help of context providers that monitor specific aspects of the operating system as well as its processes and sensors and autonomously notify the framework by means of the implementation of the observer pattern[1].

- *Integration and exploitation of dynamically discovered context sources*

  The integration of dynamically discovered context sources is a challenging issue most context-management frameworks face, especially in ubiquitous and pervasive environments. Future works in this areas should therefore focus on the investigation of additional methods for dynamic context source discovery and their integration as well as on heuristics for transforming sensorial data into qualitative context descriptions. Beneficial to the discovery and ad-hoc integration of context sources is the reliance on standard communication protocols such as HTTP and the use of well-defined and well-established Semantic Web vocabularies. One such description framework is that of *OWL-S* [MBH+04] that offers description primitives for the dynamic discovery and utilization of Web services. With the *process model* and *service grounding* parts, a sensor host can specify technical details of the interaction with a context sensor as well as the input and output data delivered by a sensor. Through the promotion of our approach we hope to motivate ubiquitous sensor hosts to apply technologies and concepts from the Semantic Web for the service descriptions of their sensors and devices.

- *Using Semantic Web-based description frameworks for the specification of context acquisition processes and workflows*

  A central aspect of the presented framework is the capability to calculate compatibilities between pairs of context providers that serve as basis for the deduction of orchestration trees and the formal specification of context acquisition workflows. However, the static structure of such context acquisition workflows is specified in an internal, implementation-specific format and as a consequence can only be interpreted by the framework. A better technological approach is to describe such context acquisition workflow specifications in a coherent and well-defined format using dedicated service composition ontologies such as OWL-S [MBH+04] that offer primitives for the description of service compositions and interactions. By conceiving every context provider as a service, a service description framework such as OWL-S can be used to represent orchestration trees as well as the context acquisition processes of the context providers being orchestrated within them and thus facilitate the coordinated invocation of context acquisition processes.

- *Adoption of standardized infrastructures and formal rule-based languages for the specification of aggregation and reasoning rules on mobile devices*

  Although the lightweight rule-based reasoner implemented in this work (see Section 4.6.2) allows for the specification of aggregation and consolidation rules in a straightforward way, they need to be formulated on the basis of a proprietary format. This approach does not only exacerbate reuse, maintenance, and the exchange of reasoning rules among reasoning systems, it also intertwines a specific reasoning engine with context aggregation and

[1] General information about the observer pattern: `http://www.oodesign.com/observer-pattern.html`

processing workflows. Therefore, a more favorable approach is the definition of reasoning rules on the basis of a well-defined and standardized rule language such as proposed in an approach targeted towards the context-aware geographical information retrieval [KRW09] in which SWRL [HPSB+04] is used as a formalism for the formulation of semantic rules. Bringing such an approach to mobile platforms requires for the development and availability of SWRL-capable interpreters and rule engines optimized for mobile platforms. Additionally, context processing can be complemented with machine learning techniques for the detection of usage patterns, as proposed by [BKL+08a, BKL08b]. To the best of our knowledge, no such engine for mobile platforms exists to date.

- *Enabling infrastructure for situation awareness*

  Since the framework proposed in this thesis provides the technical infrastructure for high-level context processing, it can serve both as a conceptual basis as well as a technical infrastructure for the deployment of more elaborated situation identification algorithms and heuristics. By collecting, representing, and disseminating contextual information on the basis of standard protocols and semantic technologies by means of RDF-based context descriptions, such information can be exchanged and incorporated between the framework as such and context consumers that aim at the augmentation and processing of acquired contextual information in order to infer on behavioral patterns that help in identifying and recognizing specific situations. The framework facilitates all relevant aspects of a mobile context processing and management architecture and serves as foundation for the systematic management and exchange of contextual information using open protocols and semantic standards. There already exists a number of approaches (cf. [ANH07, Geh08, LFWK08, SWB+08, THS09, CCMS10, YDM11]) in this area whereas none of them specifically focuses on the acquisition and local processing on a mobile device exclusively.

- *Increasing the reasoning efficiency on resource constraint devices and mobile platforms*

  With the introduction of multi-CPU and multi-GPU architectures in recent mobile devices[2], mobile reasoners can be optimized towards the parallel execution of reasoning tasks. While the mobile device's CPU handles data management tasks, reasoning processes can be delegated and distributed over GPU cores which are highly optimized for the parallel execution of calculations. To make use of such GPU-based calculations on desktop machines using conventional graphic cards, specific APIs and technologies such as *CUDA (Compute Unified Device Architecture)*[3] or *OpenCL (Open Computing Language)*[4] have been developed that allow for the parallel programming of computing-extensive tasks and their execution on a multi-GPU infrastructure. With the advent of more powerful hardware incorporated in mobile devices, the necessity of porting such interfaces for mobile architectures seems probable and paves the way for future research into the optimization of reasoning tasks.

[2]Technical specification of the NVIDIA® Tegra® 3 chip for mobile platforms: `http://www.nvidia.com/object/tegra-superchip.html`

[3]Compute Unified Device Architecture: `http://developer.nvidia.com/category/zone/cuda-zone`

[4]Open Computing Language: `http://www.khronos.org/opencl/`

Appendix A

Detailed Performance Statistics of the Replication Benchmarks

Appendix A contains the detailed results of each single benchmark conducted for evaluating the processing efficiency of local RDF data replicas as discussed in Chapter 6 using currently available RDF frameworks for mobile platforms. The RDF frameworks considered relevant for the proposed context-dependent RDF data replication infrastructure have been analyzed in Chapter 3.

TABLE A.1: Detailed results of parsing RDF data replicas on the HTC G1

		Model Size (Triples)	10	20	50	100	200	500	1000	2000	5000	10000	20000	50000	100000
Androjena	RDF/ XML	Execution Time (ms)	768,30	288,90	472,30	917,80	1.771,20	4.964,60	10.360,70	21.795,60	52.643,00	DNF	DNF	DNF	DNF
		Standard Deviation	1.906,75	2,33	2,98	9,33	9,62	96,79	376,72	2.430,40	460,09	DNF	DNF	DNF	DNF
		Triples per Second	13,02	69,23	105,86	108,96	112,92	100,71	96,52	91,76	94,98	DNF	DNF	DNF	DNF
	N3	Execution Time (ms)	157,70	223,20	337,50	655,80	1.280,30	3.398,70	7.390,50	14.051,30	DNF	DNF	DNF	DNF	DNF
		Standard Deviation	63,41	89,61	148,15	152,49	156,21	147,12	135,31	142,34	DNF	DNF	DNF	DNF	DNF
		Triples per Second	60,15	2,39	5,13	10,12	28,39	35,67	189,19	341,87	DNF	DNF	DNF	DNF	DNF
	N-Triple	Execution Time (ms)	61,50	102,00	217,80	452,60	853,80	2.226,90	5.013,80	9.623,00	DNF	DNF	DNF	DNF	DNF
		Standard Deviation	12,03	2,40	7,74	30,16	26,54	58,37	143,10	665,70	DNF	DNF	DNF	DNF	DNF
		Triples per Second	162,60	196,08	229,57	220,95	234,25	224,53	199,45	207,84	DNF	DNF	DNF	DNF	DNF
Mobile RDF	RDF/ XML	Execution Time (ms)	77,20	123,20	199,90	334,80	647,40	1.975,30	3.932,40	7.644,70	18.536,50	DNF	DNF	DNF	DNF
		Standard Deviation	129,53	162,34	250,13	298,69	308,93	253,13	254,30	261,62	269,74	DNF	DNF	DNF	DNF
		Triples per Second	35,09	34,66	73,33	31,61	85,85	108,41	304,45	219,08	318,80	DNF	DNF	DNF	DNF
µJena	N-Triple	Execution Time (ms)	36,10	792,60	1.576,00	3.496,80	1.614,60	30.357,80	84.543,40	568.003,40	DNF	DNF	DNF	DNF	DNF
		Standard Deviation	8,08	371,51	569,33	874,84	234,65	5.556,99	12.624,49	119.008,18	DNF	DNF	DNF	DNF	DNF
		Triples per Second	277,01	25,23	31,73	28,60	123,87	16,47	11,83	3,52	DNF	DNF	DNF	DNF	DNF

TABLE A.2: Detailed results of parsing RDF data replicas on the Motorola Milestone

		Model Size (Triples)	10	20	50	100	200	500	1,000	2,000	5,000	10,000	20,000	50,000	100,000
Androjena	RDF/ XML	Execution Time (ms)	335.50	138.40	199.70	314.40	550.80	1,451.60	2,963.10	6,086.40	15,050.40	31,231.90	64,873.30	DNF	DNF
		Standard Deviation	747.54	9.05	35.51	5.87	15.45	51.91	41.47	877.74	95.73	168.87	1,021.02	DNF	DNF
		Triples per Second	29.81	144.51	250.38	318.07	363.11	344.45	337.48	328.60	332.22	320.19	308.29	DNF	DNF
	N3	Execution Time (ms)	92.70	153.80	165.90	270.10	493.60	1,256.30	2,200.40	4,534.60	11,828.10	24,257.70	49,693.60	DNF	DNF
		Standard Deviation	26.38	91.86	24.06	19.86	46.87	77.31	60.07	31.40	56.54	137.50	131.33	DNF	DNF
		Triples per Second	107.87	130.04	301.39	370.23	405.19	397.99	454.46	441.05	422.72	412.24	402.47	DNF	DNF
	N-Triple	Execution Time (ms)	41.60	73.30	134.50	218.20	331.90	803.30	1,632.90	3,221.30	8,176.00	51,243.00	34,548.40	DNF	DNF
		Standard Deviation	4.38	0.82	12.46	15.80	26.73	35.82	47.92	62.29	79.69	40,861.07	535.24	DNF	DNF
		Triples per Second	240.38	272.85	371.75	458.30	602.59	622.43	612.41	620.87	611.55	195.15	578.90	DNF	DNF
Mobile RDF	RDF/ XML	Execution Time (ms)	62.90	94.80	143.80	288.30	266.20	644.30	1,291.20	2,666.00	6,671.40	13,320.30	26,291.00	DNF	DNF
		Standard Deviation	26.41	28.20	33.35	95.12	27.34	22.68	58.29	95.79	93.97	109.76	279.99	DNF	DNF
		Triples per Second	158.98	210.97	347.71	346.86	751.31	776.04	774.47	750.19	749.47	750.73	760.72	DNF	DNF
µJena	N-Triple	Execution Time (ms)	16.20	382.60	12,491.70	15,209.50	31,491.50	17,595.40	10,845.60	DNF	DNF	DNF	DNF	DNF	DNF
		Standard Deviation	8.02	137.63	24,653.70	23,229.44	41,285.81	1,996.35	1,015.30	DNF	DNF	DNF	DNF	DNF	DNF
		Triples per Second	617.28	52.27	4.00	6.57	6.35	28.42	92.20	DNF	DNF	DNF	DNF	DNF	DNF

Table A.3: Detailed results of parsing RDF data replicas on the Samsung Galaxy S I9000

		Model Size (Triples)	10	20	50	100	200	500	1,000	2,000	5,000	10,000	20,000	50,000	100,000
Androjena	RDF/ XML	Execution Time (ms)	235.00	168.00	209.00	269.00	409.60	810.00	1,540.30	3,144.70	7,559.30	18,116.60	33,256.20	115,976.80	DNF
		Standard Deviation	303.58	20.94	22.08	23.28	116.81	60.37	102.61	535.18	79.75	112.81	2,579.66	12,876.26	DNF
		Triples per Second	42.55	119.05	239.23	371.75	488.28	617.28	649.22	635.99	661.44	551.98	601.39	431.12	DNF
	N3	Execution Time (ms)	137.20	148.50	170.90	226.80	304.30	592.00	1,069.30	1,914.90	5,281.60	12,295.60	26,646.00	DNF	DNF
		Standard Deviation	59.55	19.91	22.54	22.02	25.54	48.41	91.69	114.51	90.45	256.86	1,917.43	DNF	DNF
		Triples per Second	72.89	134.68	292.57	440.92	657.25	844.59	935.19	1,044.44	946.68	813.30	750.58	DNF	DNF
	N-Triple	Execution Time (ms)	60.20	72.00	111.90	163.90	195.50	318.80	621.30	1,002.70	2,724.10	6,719.10	16,166.00	DNF	DNF
		Standard Deviation	27.70	12.10	28.63	20.51	23.61	39.85	200.39	31.53	109.69	556.76	860.41	DNF	DNF
		Triples per Second	166.11	277.78	446.83	610.13	1,023.02	1,568.38	1,609.53	1,994.61	1,835.47	1,488.29	1,237.16	DNF	DNF
Mobile RDF	RDF/ XML	Execution Time (ms)	71.60	82.00	98.10	127.30	194.10	330.00	580.80	1,060.00	2,269.20	4,789.70	11,959.30	DNF	DNF
		Standard Deviation	44.51	13.77	24.69	20.32	38.81	60.63	75.29	206.26	122.68	156.18	633.84	DNF	DNF
		Triples per Second	139.66	243.90	509.68	785.55	1,030.40	1,515.15	1,721.76	1,886.79	2,203.42	2,087.81	1,672.34	DNF	DNF
µJena	N-Triple	Execution Time (ms)	7.80	63.80	133.40	312.30	877.20	2,940.10	7,216.20	67,689.30	DNF	DNF	DNF	DNF	DNF
		Standard Deviation	13.46	21.15	39.14	46.80	99.00	402.41	955.89	17,684.88	DNF	DNF	DNF	DNF	DNF
		Triples per Second	1,282.05	313.48	374.81	320.20	228.00	170.06	138.58	29.55	DNF	DNF	DNF	DNF	DNF

Table A.4: Detailed results of parsing RDF data replicas on the Dell Streak 5

		Model Size (Triples)	10	20	50	100	200	500	1,000	2,000	5,000	10,000	20,000	50,000	100,000
Androjena	RDF/ XML	Execution Time (ms)	254.60	150.50	222.50	455.70	690.70	1,273.20	2,201.10	4,164.00	9,856.40	23,218.20	46,425.50	115,739.11	189,373.80
		Standard Deviation	360.62	5.95	3.63	61.26	84.27	82.82	70.57	660.31	106.17	364.35	2,002.97	5,546.48	3,651.79
		Triples per Second	39.28	132.89	224.72	219.44	289.56	392.71	454.32	480.31	507.28	430.70	430.80	432.01	528.06
	N3	Execution Time (ms)	119.70	104.20	217.60	335.10	536.80	1,062.00	1,584.30	2,639.50	6,700.90	15,554.40	37,430.20	DNF	DNF
		Standard Deviation	166.33	2.70	125.89	78.88	42.30	79.76	128.08	112.78	139.75	360.57	2,863.09	DNF	DNF
		Triples per Second	83.54	191.94	229.78	298.42	372.58	470.81	631.19	757.72	746.17	642.90	534.33	DNF	DNF
	N-Triple	Execution Time (ms)	32.50	37.60	77.50	167.80	315.90	612.80	963.00	1,450.00	4,180.20	8,951.50	12,667.60	29,436.90	76,487.43
		Standard Deviation	18.68	1.65	1.27	36.74	42.98	65.63	107.98	102.38	2,368.06	3,025.97	332.96	194.36	11,195.18
		Triples per Second	307.69	531.91	645.16	595.95	633.11	815.93	1,038.42	1,379.31	1,196.12	1,117.13	1,578.83	1,698.55	1,307.40
Mobile RDF	RDF/ XML	Execution Time (ms)	40.00	45.10	62.30	118.30	232.00	611.50	818.10	1,808.90	2,644.90	5,441.70	13,805.50	DNF	DNF
		Standard Deviation	11.44	1.73	1.49	1.42	1.05	97.72	36.07	689.00	103.51	157.17	1,237.12	DNF	DNF
		Triples per Second	250.00	443.46	802.57	845.31	862.07	817.66	1,222.34	1,105.64	1,890.43	1,837.66	1,448.70	DNF	DNF
µJena	N-Triple	Execution Time (ms)	7.00	87.80	187.60	463.90	275.50	2,948.90	8,422.40	53,555.40	402,844.80	DNF	DNF	DNF	DNF
		Standard Deviation	8.51	25.37	49.84	87.42	475.82	512.50	1,124.64	10,474.13	70,518.53	DNF	DNF	DNF	DNF
		Triples per Second	1,428.57	227.79	266.52	215.56	725.95	169.55	118.73	37.34	DNF	DNF	DNF	DNF	DNF

TABLE A.5: Detailed results of storing RDF data replicas on the HTC G1

	Model Size (Triples)	10	20	50	100	200	500	1000	2000	5000	10000	20000	50000	100000
Androjena	RDF/ XML													
	Execution Time (ms)	329.20	517.80	478.70	876.80	1,999.40	4,694.90	10,148.00	20,713.50	52,946.50	DNF	DNF	DNF	DNF
	Standard Deviation	114.95	11.38	5.58	20.85	29.53	100.08	274.74	740.17	448.72	DNF	DNF	DNF	DNF
	Triples per Second	30.38	38.62	104.45	114.05	100.03	106.50	98.54	96.56	94.43	DNF	DNF	DNF	DNF
	File Size (Bytes)	1,847.00	3,472.00	3,912.00	7,395.00	14,411.00	36,039.00	71,760.00	142,261.00	369,764.00	DNF	DNF	DNF	DNF
	N3													
	Execution Time (ms)	109.00	156.00	242.10	466.80	902.60	2,548.30	5,205.20	10,598.40	DNF	DNF	DNF	DNF	DNF
	Standard Deviation	50.95	1.15	2.77	14.90	10.84	24.84	51.53	264.18	DNF	DNF	DNF	DNF	DNF
	Triples per Second	91.74	128.21	206.53	214.22	221.58	196.21	192.12	188.71	DNF	DNF	DNF	DNF	DNF
	File Size (Bytes)	1,385.00	2,763.00	4,597.00	9,243.00	18,340.00	46,066.00	91,909.00	182,734.00	DNF	DNF	DNF	DNF	DNF
	N-Triple													
	Execution Time (ms)	207.70	395.30	888.70	1,779.80	3,837.00	10,177.30	20,738.90	43,593.90	DNF	DNF	DNF	DNF	DNF
	Standard Deviation	10.58	18.00	18.48	25.38	114.07	147.46	776.31	1,427.98	DNF	DNF	DNF	DNF	DNF
	Triples per Second	48.15	50.59	56.26	56.19	52.12	49.13	48.22	45.88	DNF	DNF	DNF	DNF	DNF
	File Size (Bytes)	1,232.00	2,456.00	5,804.00	11,910.00	23,613.00	59,775.00	119,616.00	237,096.00	DNF	DNF	DNF	DNF	DNF
Mobile RDF	RDF/ XML													
	Execution Time (ms)	1,462.20	1,411.80	1,458.20	1,821.20	2,329.40	3,583.90	5,868.00	10,420.40	27,577.90	DNF	DNF	DNF	DNF
	Standard Deviation	332.58	119.48	122.19	132.63	148.70	137.66	148.88	338.46	3,699.14	DNF	DNF	DNF	DNF
	Triples per Second	6.84	14.17	34.29	54.91	85.86	139.51	170.42	191.93	181.30	DNF	DNF	DNF	DNF
	File Size (Bytes)	2,678.00	4,645.00	6,253.00	11,864.00	22,905.00	56,469.00	111,862.00	221,991.00	555,956.00	DNF	DNF	DNF	DNF
jrLena	N-Triple													
	Execution Time (ms)	35.40	49.10	119.70	233.50	495.10	1,286.30	2,839.50	5,202.10	DNF	DNF	DNF	DNF	DNF
	Standard Deviation	21.56	0.88	14.53	28.52	93.90	21.10	171.08	186.73	DNF	DNF	DNF	DNF	DNF
	Triples per Second	282.49	407.33	417.71	428.27	403.96	388.71	352.17	384.46	DNF	DNF	DNF	DNF	DNF
	File Size (Bytes)	1,228.00	2,448.00	5,789.00	11,864.00	23,521.00	59,473.00	119,060.00	290,670.00	DNF	DNF	DNF	DNF	DNF

TABLE A.6: Detailed results of storing RDF data replicas on the Motorola Milestone

	Model Size (Triples)	10	20	50	100	200	500	1000	2000	5000	10000	20000	50000	100000
Androjena	Execution Time (ms)	93.60	141.80	136.10	242.10	619.40	1,383.60	2,703.30	5,694.20	14,616.30	30,694.60	66,160.30	DNF	DNF
	Standard Deviation	46.19	6.76	11.93	1.85	36.82	75.19	62.55	78.90	133.99	236.04	988.00	DNF	DNF
RDF/ XML	Triples per Second	106.84	141.04	367.38	413.05	322.89	361.38	369.92	351.23	342.08	325.79	302.30	DNF	DNF
	File Size (Bytes)	1,847.00	3,472.00	3,912.00	7,395.00	14,411.00	36,039.00	71,760.00	142,261.00	369,764.00	738,095.00	1,412,416.00	DNF	DNF
	Execution Time (ms)	33.30	44.50	68.70	128.10	246.40	892.80	1,498.50	2,923.60	7,268.50	15,833.20	33,629.10	DNF	DNF
	Standard Deviation	7.42	0.53	3.02	5.26	7.75	207.98	43.81	60.84	78.71	1,107.54	425.26	DNF	DNF
N3	Triples per Second	300.30	449.44	727.80	780.64	811.69	560.04	667.33	684.09	687.90	631.58	594.72	DNF	DNF
	File Size (Bytes)	1,385.00	2,763.00	4,597.00	9,243.00	18,340.00	46,066.00	91,909.00	182,734.00	459,150.00	917,167.00	1,832,181.00	DNF	DNF
	Execution Time (ms)	104.50	140.60	298.90	605.80	1,303.50	3,313.00	6,570.70	13,266.60	33,809.20	238,095.70	145,497.50	DNF	DNF
	Standard Deviation	20.51	11.90	11.52	35.02	52.13	59.96	50.29	84.66	465.27	160,267.56	1,818.58	DNF	DNF
N-Triple	Triples per Second	95.69	142.25	167.28	165.07	153.43	150.92	152.19	150.75	147.89	42.00	137.46	DNF	DNF
	File Size (Bytes)	1,232.00	2,456.00	5,804.00	11,910.00	23,613.00	59,775.00	119,616.00	237,096.00	593,851.00	1,187,884.00	2,379,020.00	DNF	DNF
Mobile RDF	Execution Time (ms)	76.40	72.80	166.40	280.70	334.80	661.50	1,326.40	2,887.80	7,227.20	14,993.80	26,380.20	DNF	DNF
	Standard Deviation	29.00	13.36	33.06	145.99	104.13	26.90	55.08	61.26	172.31	337.11	5,592.70	DNF	DNF
RDF/ XML	Triples per Second	130.89	274.73	300.48	356.25	597.37	755.86	753.92	692.57	691.83	666.94	758.14	DNF	DNF
	File Size (Bytes)	2,682.00	4,649.00	6,257.00	11,868.00	22,909.00	56,473.00	111,866.00	221,995.00	555,960.00	1,110,233.00	1,637,052.00	DNF	DNF
	Execution Time (ms)	10.10	18.90	40.20	75.10	157.70	450.60	843.80	DNF	DNF	DNF	DNF	DNF	DNF
	Standard Deviation	0.32	3.87	4.87	0.74	10.88	22.38	24.30	DNF	DNF	DNF	DNF	DNF	DNF
N-Triple	Triples per Second	990.10	1,058.20	1,243.78	1,331.56	1,268.23	1,109.63	1,185.11	DNF	DNF	DNF	DNF	DNF	DNF
	File Size (Bytes)	1,228.00	2,448.00	5,789.00	11,864.00	23,521.00	59,473.00	119,060.00	DNF	DNF	DNF	DNF	DNF	DNF

µJena

TABLE A.7: Detailed results of storing RDF data replicas on the Samsung Galaxy S I9000

		Model Size (Triples)	10	20	50	100	200	500	1000	2000	5000	10000	20000	50000	100000
Androjena	RDF/XML	Execution Time (ms)	105.60	116.60	97.60	144.40	317.20	735.40	1,459.40	2,715.40	7,467.90	17,042.00	32,910.50	DNF	DNF
		Standard Deviation	18.46	13.16	9.44	20.69	22.56	63.44	51.92	122.08	87.76	373.78	1,105.70	DNF	DNF
		Triples per Second	94.70	171.53	512.30	692.52	630.52	679.90	685.21	736.54	669.53	586.79	607.71	DNF	DNF
		File Size (Bytes)	1,847.00	3,472.00	3,912.00	7,395.00	14,411.00	36,039.00	71,760.00	142,261.00	369,764.00	738,095.00	1,412,416.00	DNF	DNF
	N3	Execution Time (ms)	63.90	59.50	70.60	92.60	145.20	349.20	747.10	1,566.90	3,981.70	9,306.20	19,668.50	DNF	DNF
		Standard Deviation	6.82	4.95	13.13	8.30	7.13	32.40	47.28	100.89	199.56	155.59	694.85	DNF	DNF
		Triples per Second	156.49	336.13	708.22	1,079.91	1,377.41	1,431.84	1,338.51	1,276.41	1,255.75	1,074.55	1,016.85	DNF	DNF
		File Size (Bytes)	1,385.00	2,763.00	4,597.00	9,243.00	18,340.00	46,066.00	91,909.00	182,734.00	459,150.00	917,167.00	1,832,181.00	DNF	DNF
	N-Triple	Execution Time (ms)	112.70	138.90	199.00	318.40	667.60	1,669.80	3,327.10	6,454.10	17,463.60	41,645.00	99,575.20	DNF	DNF
		Standard Deviation	19.90	22.23	9.68	31.21	91.70	64.35	257.44	380.30	197.23	708.54	5,115.08	DNF	DNF
		Triples per Second	88.73	143.99	251.26	314.07	299.58	299.44	300.56	309.88	286.31	240.13	200.85	DNF	DNF
		File Size (Bytes)	1,232.00	2,456.00	5,804.00	11,910.00	23,613.00	59,775.00	119,616.00	237,096.00	593,851.00	1,187,884.00	2,379,020.00	DNF	DNF
Mobile RDF	RDF/XML	Execution Time (ms)	67.10	73.90	75.90	83.70	276.60	352.60	655.40	1,301.30	3,096.30	7,215.40	17,323.00	DNF	DNF
		Standard Deviation	15.80	12.40	9.53	6.77	352.16	100.45	33.25	80.30	116.22	298.55	1,200.68	DNF	DNF
		Triples per Second	149.03	270.64	658.76	1,194.74	723.07	1,418.04	1,525.79	1,536.92	1,614.83	1,385.92	1,154.53	DNF	DNF
		File Size (Bytes)	2,672.00	4,639.00	6,247.00	11,858.00	22,899.00	56,463.00	111,856.00	221,985.00	555,950.00	1,110,223.00	2,217,572.00	DNF	DNF
μJena	N-Triple	Execution Time (ms)	31.60	36.70	69.10	123.30	270.20	642.50	1,344.30	2,618.50	6,478.40	DNF	DNF	DNF	DNF
		Standard Deviation	7.96	2.75	5.15	15.24	14.45	17.68	22.56	42.07	205.29	DNF	DNF	DNF	DNF
		Triples per Second	316.46	544.96	723.59	811.03	740.19	778.21	743.88	763.80	771.80	DNF	DNF	DNF	DNF
		File Size (Bytes)	1,228.00	2,448.00	5,789.00	11,864.00	23,521.00	59,473.00	119,060.00	290,670.00	731,670.00	DNF	DNF	DNF	DNF

TABLE A.8: Detailed results of storing RDF data replicas on the Dell Streak 5

	Model Size (Triples)	10	20	50	100	200	500	1000	2000	5000	10000	20000	50000	100000
Androjena	Execution Time (ms)	272.20	293.10	244.20	273.60	355.50	759.80	1,494.20	2,856.90	7,876.50	18,988.10	39,738.20	118,246.57	DNF
	Standard Deviation	90.45	27.06	10.49	96.33	47.90	53.13	68.66	220.98	147.91	255.13	2,123.01	668.74	DNF
RDF/ XML	Triples per Second	36.74	68.24	204.75	365.50	562.59	658.07	669.25	700.06	634.80	526.65	503.29	422.85	DNF
	File Size (Bytes)	1,847.00	3,472.00	3,912.00	7,395.00	14,411.00	36,039.00	71,760.00	142,261.00	369,764.00	738,095.00	1,412,416.00	3,542,782.00	DNF
	Execution Time (ms)	83.30	98.80	189.90	271.60	237.20	401.20	880.10	1,965.20	5,033.90	11,618.70	26,417.60	DNF	DNF
	Standard Deviation	26.46	4.96	87.33	61.70	107.56	29.55	46.95	142.47	31.11	80.47	883.82	DNF	DNF
N3	Triples per Second	120.05	202.43	263.30	368.19	843.17	1,246.26	1,136.23	1,017.71	993.27	860.68	757.07	DNF	DNF
	File Size (Bytes)	1,385.00	2,763.00	4,597.00	9,243.00	18,340.00	46,066.00	91,909.00	182,734.00	459,150.00	917,167.00	1,832,181.00	DNF	DNF
	Execution Time (ms)	116.30	198.30	439.10	572.10	746.90	1,430.10	2,797.70	5,607.20	18,488.20	42,104.90	73,566.10	193,681.60	DNF
	Standard Deviation	12.24	1.06	16.97	147.99	186.19	61.37	62.28	73.61	6,589.79	911.99	5,021.44	2,364.33	DNF
N-Triple	Triples per Second	85.98	100.86	113.87	174.79	267.77	349.63	357.44	356.68	270.44	237.50	271.86	258.16	DNF
	File Size (Bytes)	1,232.00	2,456.00	5,804.00	11,910.00	23,613.00	59,775.00	119,616.00	237,096.00	593,851.00	1,187,884.00	2,379,020.00	5,929,186.00	DNF
Mobile RDF	Execution Time (ms)	77.20	84.60	109.20	183.80	332.20	330.20	615.40	1,488.70	2,797.00	6,855.00	18,717.00	DNF	DNF
	Standard Deviation	32.39	8.87	5.79	13.16	65.32	66.97	10.31	982.99	93.87	238.32	1,855.15	DNF	DNF
RDF/ XML	Triples per Second	129.53	236.41	457.88	544.07	602.05	1,514.23	1,624.96	1,343.45	1,787.63	1,458.79	1,068.55	DNF	DNF
	File Size (Bytes)	2,678.00	4,645.00	6,253.00	11,864.00	22,905.00	56,469.00	111,862.00	221,991.00	555,956.00	1,110,229.00	2,217,578.00	DNF	DNF
	Execution Time (ms)	25.60	24.00	47.60	99.50	305.10	555.60	1,162.60	2,309.80	5,459.90	DNF	DNF	DNF	DNF
	Standard Deviation	9.98	4.06	1.35	32.86	201.52	34.80	33.18	140.37	62.20	DNF	DNF	DNF	DNF
N-Triple	Triples per Second	390.63	833.33	1,050.42	1,005.03	655.52	899.93	860.14	865.88	915.77	DNF	DNF	DNF	DNF
	File Size (Bytes)	1,228.00	2,448.00	5,789.00	11,864.00	23,521.00	59,473.00	119,060.00	290,670.00	731,670.00	DNF	DNF	DNF	DNF
μJena														

TABLE A.9: Detailed results of adding data to RDF data replicas using the Androjena framework on the HTC G1

Model Size	Add 1	Std. Dev.	Triples/Sec	Add 10	Std. Dev.	Triples/Sec	Add 100	Std. Dev.	Triples/Sec	Add 500	Std. Dev.	Triples/Sec	Add 1000	Std. Dev.	Triples/Sec
10	1.60	0.52	625.00	16.90	4.65	591.72	167.80	56.07	595.95	944.50	117.21	529.38	1,614.60	137.40	619.35
20	1.50	0.53	666.67	14.60	0.97	684.93	144.70	14.94	691.09	958.90	8.77	521.43	1,703.30	30.16	587.10
50	1.50	0.53	666.67	14.30	0.48	699.30	138.80	1.40	720.46	724.70	7.02	689.94	1,702.40	11.07	587.41
100	1.40	0.52	714.29	14.60	0.52	684.93	142.80	1.55	700.28	985.20	21.05	507.51	1,715.40	13.79	582.95
200	1.30	0.48	769.23	14.50	0.53	689.66	141.40	1.43	707.21	738.90	17.59	676.68	1,736.10	12.91	576.00
500	1.40	0.52	714.29	14.70	0.48	680.27	143.20	2.25	698.32	750.30	81.58	666.40	1,875.80	148.80	533.11
1,000	1.40	0.52	714.29	14.90	0.57	671.14	146.10	2.18	684.46	757.80	5.90	659.80	1,671.10	138.86	598.41
2,000	1.60	0.52	625.00	14.80	0.63	675.68	159.20	35.18	628.14	781.10	78.08	640.12	1,998.10	370.42	500.48
5,000	1.40	0.52	714.29	15.30	0.48	653.59	256.80	237.02	389.41	843.70	246.54	592.63	1,875.50	256.10	533.19
10,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
20,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
50,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
100,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
			689.97			670.14			646.15			609.32			568.67

TABLE A.10: Detailed results of adding data to RDF data replicas using the Androjena framework on the Motorola Milestone

Model Size	Add 1	Std. Dev.	Triples/Sec	Add 10	Std. Dev.	Triples/Sec	Add 100	Std. Dev.	Triples/Sec	Add 500	Std. Dev.	Triples/Sec	Add 1000	Std. Dev.	Triples/Sec
10	0.60	0.70	1,666.67	5.30	4.11	1,886.79	36.90	0.88	2,710.03	200.70	30.54	2,491.28	527.10	40.99	1,897.17
20	0.20	0.42	5,000.00	4.00	0.47	2,500.00	37.90	4.63	2,638.52	191.30	5.38	2,613.70	411.80	40.12	2,428.36
50	0.90	0.32	1,111.11	3.50	0.53	2,857.14	36.90	0.99	2,710.03	365.30	145.52	1,368.74	396.20	11.34	2,523.98
100	0.50	0.53	2,000.00	3.90	0.32	2,564.10	155.10	2.88	644.75	192.80	3.33	2,593.36	433.00	45.66	2,309.47
200	0.50	0.53	2,000.00	4.10	0.32	2,439.02	38.50	5.82	2,597.40	193.50	3.75	2,583.98	504.90	3.78	1,980.59
500	0.40	0.52	2,500.00	3.80	0.42	2,631.58	38.10	2.81	2,624.67	319.60	4.45	1,564.46	550.70	58.06	1,815.87
1,000	0.30	0.48	3,333.33	3.90	0.32	2,564.10	37.70	0.48	2,652.52	201.60	0.97	2,480.16	529.00	22.24	1,890.36
2,000	0.40	0.52	2,500.00	4.20	0.63	2,380.95	39.60	0.52	2,525.25	208.30	42.76	2,400.38	570.50	64.01	1,752.85
5,000	0.60	0.52	1,666.67	3.90	0.32	2,564.10	37.20	1.03	2,688.17	200.70	21.54	2,491.28	398.60	16.93	2,508.78
10,000	0.50	0.53	2,000.00	4.00	0.00	2,500.00	36.50	0.53	2,739.73	217.50	73.68	2,298.85	413.20	78.66	2,420.14
20,000	0.14	0.38	7,000.00	3.80	0.42	2,631.58	40.29	6.95	2,482.27	326.50	198.86	1,531.39	424.30	100.99	2,356.82
50,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
100,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
			2,797.98			2,501.76			2,455.76			2,219.78			2,171.31

Table A.11: Detailed results of adding data to RDF data replicas using the Androjena framework on the Samsung Galaxy S I9000

Model Size	Add 1	Std. Dev.	Triples/Sec	Add 10	Std. Dev.	Triples/Sec	Add 100	Std. Dev.	Triples/Sec	Add 500	Std. Dev.	Triples/Sec	Add 1000	Std. Dev.	Triples/Sec
10	0.30	0.48	3,333.33	2.60	2.27	3,846.15	20.60	18.82	4,854.37	94.70	31.00	5,279.83	177.80	31.37	5,624.30
20	0.30	0.79	3,333.33	1.40	0.52	7,142.86	14.40	2.46	6,944.44	135.00	5.89	3,703.70	215.40	7.56	4,642.53
50	0.30	0.48	3,333.33	1.80	1.23	5,555.56	15.10	2.02	6,622.52	82.60	6.43	6,053.27	227.90	27.04	4,387.89
100	0.30	0.48	3,333.33	1.70	1.25	5,882.35	16.00	1.76	6,250.00	136.90	6.23	3,652.30	226.90	17.68	4,407.23
200	0.20	0.42	5,000.00	1.80	1.23	5,555.56	15.60	2.41	6,410.26	127.10	66.04	3,933.91	229.20	16.82	4,363.00
500	0.30	0.48	3,333.33	2.00	1.15	5,000.00	16.50	1.65	6,060.61	86.30	7.02	5,793.74	246.50	38.60	4,056.80
1,000	0.30	0.48	3,333.33	1.60	0.52	6,250.00	16.30	1.42	6,134.97	105.50	59.71	4,739.34	188.00	41.79	5,319.15
2,000	0.20	0.42	5,000.00	2.00	1.49	5,000.00	19.30	4.74	5,181.35	84.60	4.30	5,910.17	288.10	80.58	3,471.02
5,000	0.60	0.97	1,666.67	1.70	0.48	5,882.35	16.10	4.01	6,211.18	96.20	36.50	5,197.51	294.40	35.03	3,396.74
10,000	0.20	0.42	5,000.00	1.60	0.52	6,250.00	73.40	91.34	1,362.40	83.50	3.69	5,988.02	302.30	106.11	3,307.97
20,000	0.40	0.52	2,500.00	1.90	0.32	5,263.16	17.10	5.11	5,847.95	83.10	10.85	6,016.85	179.00	57.42	5,586.59
50,000	0.20	0.42	5,000.00	2.00	0.67	5,000.00	15.14	0.38	6,603.77	206.70	185.94	2,418.96	533.50	248.22	1,874.41
100,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
			3,680.56			5,552.33			5,706.98			4,890.63			4,203.14

Table A.12: Detailed results of adding data to RDF data replicas using the Androjena framework on the Dell Streak 5

Model Size	Add 1	Std. Dev.	Triples/Sec	Add 10	Std. Dev.	Triples/Sec	Add 100	Std. Dev.	Triples/Sec	Add 500	Std. Dev.	Triples/Sec	Add 1000	Std. Dev.	Triples/Sec
10	0.70	0.48	1,428.57	7.00	3.74	1,428.57	53.00	21.30	1,886.79	233.70	55.53	2,139.50	440.30	165.33	2,271.18
20	0.30	0.42	3,333.33	5.70	1.89	1,754.39	27.00	16.66	3,703.70	150.70	29.08	3,317.85	205.60	0.84	4,863.81
50	0.30	0.48	3,333.33	5.10	1.66	1,960.78	26.10	16.98	3,831.42	69.90	0.99	7,153.08	208.20	1.48	4,803.07
100	0.10	0.32	10,000.00	1.20	0.42	8,333.33	13.80	1.62	7,246.38	153.00	31.44	3,267.97	219.60	23.71	4,553.73
200	0.30	0.48	3,333.33	1.60	0.70	6,250.00	13.20	0.79	7,575.76	76.90	20.09	6,501.95	214.90	0.88	4,653.33
500	0.10	0.32	10,000.00	1.60	0.52	6,250.00	14.70	2.41	6,802.72	71.50	1.72	6,993.01	262.80	107.86	3,805.18
1,000	0.20	0.42	5,000.00	1.60	0.52	6,250.00	13.90	0.32	7,194.24	87.20	23.49	5,733.94	168.10	45.64	5,948.84
2,000	0.80	0.92	1,250.00	1.60	0.97	6,250.00	18.00	7.27	5,555.56	75.00	4.50	6,666.67	254.00	42.18	3,937.01
5,000	0.20	0.42	5,000.00	1.60	0.52	6,250.00	13.50	0.71	7,407.41	83.40	45.54	5,995.20	295.00	4.29	3,389.83
10,000	0.10	0.32	10,000.00	1.70	0.48	5,882.35	69.00	118.09	1,449.28	76.50	10.58	6,535.95	334.70	107.14	2,987.75
20,000	0.10	0.32	10,000.00	2.00	0.82	5,000.00	18.10	7.36	5,524.86	81.80	3.79	6,112.47	223.30	108.36	4,478.28
50,000	0.10	0.32	10,000.00	4.30	1.77	2,325.58	17.50	6.94	5,050.51	148.00	141.52	3,378.38	181.60	67.33	5,506.61
100,000	DNF	DNF	DNF	2.40	1.07	4,166.67	23.36	DNF	DNF	96.63	DNF	DNF	496.80	244.38	2,012.88
			6,056.55			4,777.05			5,269.05			5,316.33			4,093.19

TABLE A.13: Detailed results of adding data to RDF data replicas using the μJena framework on the HTC G1

Model Size	Add 1	Std. Dev.	Triples/Sec	Add 10	Std. Dev.	Triples/Sec	Add 100	Std. Dev.	Triples/Sec	Add 500	Std. Dev.	Triples/Sec	Add 1000	Std. Dev.	Triples/Sec
10	3.40	0.52	294.12	65.00	10.93	153.85	3,642.50	475.52	27.45	82,054.60	6,875.91	6.09	2,892,062.40	8,035,947.32	0.35
20	4.10	0.32	243.90	67.30	0.82	148.59	3,656.80	55.97	27.35	82,041.80	1,011.25	6.09	341,811.20	4,466.82	2.93
50	4.30	0.48	232.56	74.00	10.58	135.14	3,710.80	141.64	26.95	81,047.50	959.62	6.17	344,484.70	11,401.40	2.90
100	5.40	0.52	185.19	93.10	36.88	107.41	3,767.00	127.22	26.55	80,655.90	1,745.70	6.20	347,916.00	16,697.74	2.87
200	14.70	0.82	68.03	192.00	44.32	52.08	5,182.40	809.84	19.30	92,197.20	10,796.89	5.42	379,119.10	31,998.16	2.64
500	23.10	0.74	43.29	268.20	4.47	37.29	5,769.60	166.19	17.33	92,682.60	2,913.83	5.39	381,299.20	18,333.39	2.62
1,000	38.90	2.13	25.71	429.00	62.28	23.31	7,559.80	227.09	13.23	105,992.30	1,335.51	4.72	410,492.50	6,897.45	2.44
2,000	370.10	15.03	2.70	4,072.00	902.84	2.46	43,908.90	9,510.07	2.28	1,039,865.00	2,451,383.76	0.48	735,100.30	74,386.14	1.36
5,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
10,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
20,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
50,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
100,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
			136.94			82.51			20.05			5.07			2.26

TABLE A.14: Detailed results of adding data to RDF data replicas using the μJena framework on the Motorola Milestone

Model Size	Add 1	Std. Dev.	Triples/Sec	Add 10	Std. Dev.	Triples/Sec	Add 100	Std. Dev.	Triples/Sec	Add 500	Std. Dev.	Triples/Sec	Add 1000	Std. Dev.	Triples/Sec
10	1.90	1.52	526.32	23.90	2.51	418.41	12,980.80	24,531.26	7.70	199,166.70	120,513.92	2.51	959,218.60	554,184.63	1.04
20	1.40	0.52	714.29	25.40	1.26	393.70	17,623.10	26,639.39	5.67	271,330.00	44,026.63	1.84	1,229,653.80	76,583.29	0.81
50	1.80	0.63	555.56	26.30	0.95	380.23	24,810.40	30,409.30	4.03	302,407.10	43,841.51	1.65	1,298,560.70	114,414.09	0.77
100	2.10	0.32	476.19	2,922.70	9,139.79	3.42	7,906.00	13,970.69	12.65	305,836.30	52,140.83	1.63	1,302,172.00	114,492.45	0.77
200	3.00	0.00	333.33	42.40	6.42	235.85	6,830.20	16,956.64	14.64	131,551.50	136,514.47	3.80	470,193.00	565,242.70	2.13
500	5.80	0.42	172.41	68.90	2.56	145.14	1,753.60	54.81	57.03	32,515.00	131.70	15.38	136,447.90	3,592.99	7.33
1,000	83.40	39.90	11.99	279.90	57.72	35.73	4,077.10	604.46	24.53	44,640.60	4,840.14	11.20	160,415.80	11,647.86	6.23
2,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
5,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
10,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
20,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
50,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
100,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
			398.58			230.35			18.04			5.43			2.73

Table A.15: Detailed results of adding data to RDF data replicas using the μJena framework on the Samsung Galaxy S I9000

Model Size	Add 1	Std. Dev.	Triples/Sec	Add 10	Std. Dev.	Triples/Sec	Add 100	Std. Dev.	Triples/Sec	Add 500	Std. Dev.	Triples/Sec	Add 1000	Std. Dev.	Triples/Sec
10	0.20	0.42	5,000.00	8.40	11.47	1,190.48	302.40	191.00	330.69	6,336.30	807.39	78.91	34,721.30	11,572.19	28.80
20	0.40	0.52	2,500.00	5.10	0.32	1,960.78	237.90	3.57	420.34	5,911.10	58.03	84.59	26,614.10	292.43	37.57
50	0.50	0.53	2,000.00	5.20	0.42	1,923.08	270.10	3.48	370.23	5,931.50	55.30	84.30	28,572.50	2,673.60	35.00
100	0.40	0.52	2,500.00	5.80	0.42	1,724.14	243.60	2.72	410.51	6,039.20	20.68	82.79	28,094.90	2,935.94	35.59
200	0.80	0.42	1,250.00	7.40	0.52	1,351.35	264.40	9.58	378.21	6,176.10	52.61	80.96	27,566.20	969.46	36.28
500	1.00	0.00	1,000.00	12.20	0.42	819.67	305.80	2.25	327.01	6,588.00	25.59	75.90	31,532.00	825.51	31.71
1,000	2.00	0.00	500.00	49.20	20.23	203.25	421.50	3.06	237.25	7,289.70	128.28	68.59	32,989.00	4,560.55	30.31
2,000	11.30	0.67	88.50	156.60	47.22	63.86	1,433.00	43.52	69.78	11,919.60	45.59	41.95	38,451.20	550.75	26.01
5,000	35.20	18.56	28.41	392.50	44.86	25.48	3,267.80	45.36	30.60	20,892.70	53.07	23.93	55,496.50	109.87	18.02
10,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
20,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
50,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
100,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
			1,651.88			1,029.12			286.07			69.10			31.03

Table A.16: Detailed results of adding data to RDF data replicas using the μJena framework on the Dell Streak 5

Model Size	Add 1	Std. Dev.	Triples/Sec	Add 10	Std. Dev.	Triples/Sec	Add 100	Std. Dev.	Triples/Sec	Add 500	Std. Dev.	Triples/Sec	Add 1000	Std. Dev.	Triples/Sec
10	0.50	0.53	2,000.00	10.60	15.96	943.40	436.20	516.49	229.25	7,865.80	2,392.46	63.57	39,062.80	1,066.65	25.60
20	0.60	0.52	1,666.67	6.00	0.67	1,666.67	260.10	2.38	384.47	7,151.70	70.58	69.91	33,325.60	621.70	30.01
50	0.50	0.53	2,000.00	6.00	0.00	1,666.67	306.20	12.00	326.58	7,100.90	64.97	70.41	33,206.00	3,004.05	30.12
100	0.50	0.53	2,000.00	7.10	0.32	1,408.45	280.60	22.17	356.38	7,247.60	63.28	68.99	34,525.30	3,151.30	28.96
200	1.20	0.42	833.33	36.60	68.31	273.22	568.20	564.09	175.99	8,732.50	3,064.49	57.26	36,188.80	2,042.97	27.63
500	2.00	0.00	500.00	26.20	14.75	381.68	443.90	37.83	225.28	8,478.10	79.68	58.98	38,629.40	1,066.51	25.89
1,000	6.70	12.06	149.25	52.20	26.79	191.57	596.10	22.58	167.76	9,484.70	227.39	52.72	41,863.80	3,444.85	23.89
2,000	44.10	17.15	22.68	435.30	104.57	22.97	4,532.80	908.75	22.06	28,716.10	4,469.74	17.41	73,840.60	6,788.46	13.54
5,000	79.80	43.39	12.53	743.50	17.15	13.45	6,704.80	76.96	14.91	39,259.90	125.80	12.74	95,123.60	199.67	10.51
10,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
20,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
50,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
100,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
			1,020.50			729.79			211.41			52.44			24.02

TABLE A.17: Detailed results of adding data to RDF data replicas using the Mobile RDF framework on the HTC G1

Model Size	Add 1	Std. Dev.	Triples/Sec	Add 10	Std. Dev.	Triples/Sec	Add 100	Std. Dev.	Triples/Sec	Add 500	Std. Dev.	Triples/Sec	Add 1000	Std. Dev.	Triples/Sec
10	1.10	0.32	909.09	10.20	1.40	980.39	217.20	58.88	460.41	697.60	93.31	716.74	1,651.60	137.87	605.47
20	0.90	0.32	1,111.11	10.70	2.95	934.58	223.90	21.68	446.63	691.10	95.14	723.48	1,610.90	133.16	620.77
50	0.90	0.32	1,111.11	9.60	0.52	1,041.67	98.40	1.07	1,016.26	865.70	135.57	577.57	1,518.30	102.31	658.63
100	0.90	0.32	1,111.11	11.50	3.57	869.57	103.20	19.40	968.99	711.50	109.50	702.74	1,562.20	125.44	640.12
200	1.00	0.00	1,000.00	10.50	0.97	952.38	253.90	12.66	393.86	819.10	53.53	610.43	1,739.20	124.91	574.98
500	0.90	0.32	1,111.11	13.20	5.31	757.58	101.00	3.30	990.10	786.70	106.04	635.57	1,552.70	104.53	644.04
1,000	1.00	0.00	1,000.00	10.80	1.55	925.93	154.80	115.32	645.99	779.80	146.64	641.19	1,802.30	175.98	554.85
2,000	2.10	3.48	476.19	10.30	0.48	970.87	111.40	31.93	897.67	731.60	125.44	683.43	1,468.60	163.53	680.92
5,000	1.00	0.00	1,000.00	13.00	7.75	769.23	112.30	30.96	890.47	803.30	250.99	622.43	1,635.00	246.26	611.62
10,000	549.50	407.79	1.82	769.10	352.49	13.00	910.40	341.92	109.84	1,365.50	184.40	366.17	1,739.40	366.84	574.91
20,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
50,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
100,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
			981.08			911.35			682.02			627.98			616.63

TABLE A.18: Detailed results of adding data to RDF data replicas using the Mobile RDF framework on the Motorola Milestone

Model Size	Add 1	Std. Dev.	Triples/Sec	Add 10	Std. Dev.	Triples/Sec	Add 100	Std. Dev.	Triples/Sec	Add 500	Std. Dev.	Triples/Sec	Add 1000	Std. Dev.	Triples/Sec
10	0.40	0.52	2,500.00	2.60	0.52	3,846.15	30.40	8.15	3,289.47	293.00	45.22	1,706.48	462.10	35.67	2,164.03
20	0.10	0.32	10,000.00	2.70	0.48	3,703.70	26.70	0.67	3,745.32	269.10	19.18	1,858.05	533.80	50.93	1,873.36
50	0.50	0.53	2,000.00	2.40	0.52	4,166.67	26.80	0.42	3,731.34	232.60	32.27	2,149.61	517.10	44.59	1,933.86
100	0.40	0.52	2,500.00	2.60	0.52	3,846.15	27.90	2.18	3,584.23	278.60	26.79	1,794.69	467.60	40.14	2,138.58
200	0.40	0.52	2,500.00	2.40	0.52	4,166.67	87.10	22.88	1,148.11	274.70	23.43	1,820.17	501.00	2.83	1,996.01
500	0.30	0.48	3,333.33	3.00	0.00	3,333.33	30.90	6.01	3,236.25	237.00	20.78	2,109.70	396.80	19.97	2,520.16
1,000	0.20	0.42	5,000.00	3.10	0.32	3,225.81	32.20	6.32	3,105.59	164.30	29.29	3,043.21	514.80	36.94	1,942.50
2,000	0.40	0.52	2,500.00	2.90	0.32	3,448.28	30.80	3.61	3,246.75	275.70	59.58	1,813.57	485.20	72.01	2,061.01
5,000	0.20	0.42	5,000.00	5.10	7.36	1,960.78	33.30	5.14	3,003.00	160.60	23.14	3,113.33	551.20	28.77	1,814.22
10,000	0.30	0.48	3,333.33	3.00	0.00	3,333.33	33.80	4.24	2,958.58	163.20	4.10	3,063.73	361.10	106.14	2,769.32
20,000	DNF	DNF	DNF	2.80	0.45	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
50,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
100,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
			3,866.67			3,503.09			3,104.86			2,247.25			2,121.31

Table A.19: Detailed results of adding data to RDF data replicas using the Mobile RDF framework on the Samsung Galaxy S I9000

Model Size	Add 1	Std. Dev.	Triples/Sec	Add 10	Std. Dev.	Triples/Sec	Add 100	Std. Dev.	Triples/Sec	Add 500	Std. Dev.	Triples/Sec	Add 1000	Std. Dev.	Triples/Sec
10	0.10	0.32	10,000.00	2.20	2.49	4,545.45	14.60	7.59	6,849.32	128.60	13.19	3,888.02	225.90	33.50	4,426.74
20	0.20	0.42	5,000.00	1.60	1.26	6,250.00	10.80	2.97	9,259.26	116.00	8.18	4,310.34	207.70	15.63	4,814.64
50	0.10	0.32	10,000.00	1.50	1.27	6,666.67	10.50	1.43	9,523.81	87.90	11.87	5,688.28	237.70	29.14	4,206.98
100	0.10	0.00	10,000.00	1.00	0.00	10,000.00	9.70	2.36	10,309.28	115.10	15.09	4,344.05	235.20	30.45	4,251.70
200	0.10	0.00	10,000.00	0.70	0.48	14,285.71	12.30	2.41	8,130.08	93.90	15.42	5,324.81	206.30	12.48	4,847.31
500	0.10	0.32	10,000.00	0.70	0.48	14,285.71	8.70	1.57	11,494.25	121.00	15.79	4,132.23	275.30	141.77	3,632.40
1,000	0.10	0.32	10,000.00	1.60	1.26	6,250.00	49.80	7.16	2,008.03	125.70	2.36	3,977.72	271.20	65.67	3,687.32
2,000	0.10	0.00	10,000.00	1.10	0.74	9,090.91	13.60	14.92	7,352.94	107.50	26.16	4,651.16	195.00	44.40	5,128.21
5,000	0.10	0.32	10,000.00	1.00	0.00	10,000.00	8.80	0.63	11,363.64	68.80	52.48	7,267.44	245.30	52.33	4,076.64
10,000	0.20	0.42	5,000.00	1.10	0.32	9,090.91	25.60	49.76	3,906.25	66.50	51.93	7,518.80	287.90	38.20	3,473.43
20,000	0.60	1.26	1,666.67	1.10	0.57	9,090.91	50.10	127.87	1,996.01	241.80	199.97	2,067.82	396.50	102.04	2,522.07
50,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
100,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
			8,333.33			9,050.57			7,472.08			4,833.70			4,097.04

Table A.20: Detailed results of adding data to RDF data replicas using the Mobile RDF framework on the Dell Streak 5

Model Size	Add 1	Std. Dev.	Triples/Sec	Add 10	Std. Dev.	Triples/Sec	Add 100	Std. Dev.	Triples/Sec	Add 500	Std. Dev.	Triples/Sec	Add 1000	Std. Dev.	Triples/Sec
10	0.10	0.32	10,000.00	5.90	7.85	1,694.92	40.50	42.15	2,469.14	317.10	88.46	1,576.79	398.90	219.13	2,506.89
20	0.40	0.52	2,500.00	3.50	0.97	2,857.14	29.10	0.57	3,436.43	356.10	99.27	1,404.10	415.20	255.77	2,408.48
50	0.30	0.48	3,333.33	3.50	0.53	2,857.14	29.40	0.52	3,401.36	200.10	87.35	2,498.75	266.80	30.41	3,748.13
100	0.10	0.32	10,000.00	4.00	0.67	2,500.00	28.00	6.32	3,571.43	131.40	14.89	3,805.18	243.00	0.82	4,115.23
200	0.30	0.48	3,333.33	1.20	1.32	8,333.33	11.80	3.33	8,474.58	92.10	15.21	5,428.88	213.30	0.67	4,688.23
500	0.20	0.42	5,000.00	0.80	0.42	12,500.00	7.80	0.42	12,820.51	129.30	0.67	3,866.98	255.20	0.79	3,918.50
1,000	0.10	0.32	10,000.00	1.00	0.00	10,000.00	54.20	0.63	1,845.02	136.00	0.82	3,676.47	256.10	27.73	3,904.72
2,000	0.20	0.42	5,000.00	1.60	1.35	6,250.00	8.90	2.51	11,235.96	147.50	49.81	3,389.83	184.50	29.56	5,420.05
5,000	0.10	0.32	10,000.00	0.90	0.32	11,111.11	8.20	0.42	12,195.12	57.00	41.12	8,771.93	247.60	41.86	4,038.77
10,000	0.10	0.32	10,000.00	1.00	0.00	10,000.00	8.20	0.42	12,195.12	44.20	0.63	11,312.22	327.00	5.48	3,058.10
20,000	0.30	0.48	3,333.33	1.20	0.42	8,333.33	48.40	120.42	2,066.12	308.50	224.13	1,620.75	512.50	151.89	1,951.22
50,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
100,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
			6,590.91			6,948.82			6,700.98			4,304.72			3,614.39

Table A.21: Detailed results of removing data from RDF data replicas using the Androjena framework on the HTC G1

Model Size	Remove 1	Std. Dev.	Triples/Sec	Remove 10	Std. Dev.	Triples/Sec	Remove 100	Std. Dev.	Triples/Sec	Remove 500	Std. Dev.	Triples/Sec	Remove 1000	Std. Dev.	Triples/Sec
10	1.20	0.42	833.33	12.10	0.32	826.45	129.60	28.89	771.60	926.60	144.55	539.61	1,848.00	173.55	541.13
20	1.10	0.32	909.09	11.50	0.53	869.57	117.90	1.37	848.18	948.90	134.02	526.93	1,873.20	164.07	533.85
50	1.10	0.32	909.09	11.90	0.88	840.34	134.80	37.49	741.84	953.40	129.92	524.44	1,864.10	148.58	536.45
100	1.10	0.32	909.09	11.90	0.57	840.34	128.10	34.78	780.64	703.70	67.68	710.53	1,615.70	145.17	618.93
200	1.20	0.42	833.33	11.10	0.32	900.90	381.20	3.05	262.33	666.30	11.36	750.41	1,745.30	17.88	572.97
500	1.20	0.42	833.33	19.90	6.17	502.51	117.90	1.20	848.18	667.70	4.81	748.84	1,750.50	20.07	571.27
1,000	1.00	0.00	1,000.00	11.80	0.42	847.46	116.40	0.97	859.11	672.60	3.27	743.38	1,533.50	27.62	652.10
2,000	1.20	0.42	833.33	11.80	0.63	847.46	116.60	1.07	857.63	667.20	6.23	749.40	1,789.10	94.80	558.94
5,000	1.30	0.48	769.23	24.30	7.82	411.52	126.70	18.76	789.27	689.40	7.65	725.27	1,630.30	131.09	613.38
10,000	1.30	0.48	769.23	12.20	0.42	819.67	331.20	341.33	301.93	724.10	8.90	690.51	1,863.20	283.69	536.71
20,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
50,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
100,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
			859.91			770.62			706.07			670.93			573.57

Table A.22: Detailed results of removing data from RDF data replicas using the Androjena framework on the Motorola Milestone

Model Size	Remove 1	Std. Dev.	Triples/Sec	Remove 10	Std. Dev.	Triples/Sec	Remove 100	Std. Dev.	Triples/Sec	Remove 500	Std. Dev.	Triples/Sec	Remove 1000	Std. Dev.	Triples/Sec
10	0.50	0.53	2,000.00	4.40	1.71	2,272.73	33.10	0.74	3,021.15	192.50	18.00	2,597.40	476.10	68.30	2,100.40
20	0.50	0.53	2,000.00	3.60	0.52	2,777.78	33.20	0.63	3,012.05	198.30	26.46	2,521.43	435.50	7.85	2,296.21
50	0.40	0.52	2,500.00	3.50	0.71	2,857.14	34.20	4.18	2,923.98	200.80	34.09	2,490.04	555.80	44.63	1,799.21
100	0.30	0.48	3,333.33	3.40	0.52	2,941.18	32.90	2.18	3,039.51	296.10	27.38	1,688.62	559.10	47.68	1,788.59
200	0.60	0.52	1,666.67	3.50	0.53	2,857.14	32.60	0.52	3,067.48	190.80	6.78	2,620.55	460.50	35.89	2,171.55
500	0.70	0.48	1,428.57	3.30	0.48	3,030.30	35.20	8.78	2,840.91	210.10	43.66	2,379.82	570.50	34.55	1,752.85
1,000	0.50	0.53	2,000.00	3.70	0.48	2,702.70	33.10	0.57	3,021.15	201.20	26.86	2,485.09	442.40	12.34	2,260.40
2,000	0.30	0.48	3,333.33	3.90	0.32	2,564.10	36.20	6.29	2,762.43	191.70	2.58	2,608.24	460.60	36.93	2,171.08
5,000	0.40	0.52	2,500.00	3.70	0.48	2,702.70	33.60	0.70	2,976.19	198.40	5.62	2,520.16	459.90	10.42	2,174.39
10,000	0.60	0.52	1,666.67	3.80	0.42	2,631.58	34.40	2.27	2,906.98	329.70	91.95	1,516.53	620.90	84.97	1,610.57
20,000	0.40	0.52	2,500.00	3.40	0.52	2,941.18	92.30	119.09	1,083.42	192.20	2.90	2,601.46	511.60	117.81	1,954.65
50,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
100,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
			2,242.86			2,733.74			2,957.18			2,342.79			2,012.52

Table A.23: Detailed results of removing data from RDF data replicas using the Androjena framework on the Samsung Galaxy S I9000

Model Size	Remove 1	Std. Dev.	Triples/Sec	Remove 10	Std. Dev.	Triples/Sec	Remove 100	Std. Dev.	Triples/Sec	Remove 500	Std. Dev.	Triples/Sec	Remove 1000	Std. Dev.	Triples/Sec
10	0.30	0.48	3,333.33	3.30	2.06	3,030.30	18.50	11.17	5,405.41	112.20	10.84	4,456.33	196.00	8.87	5,102.04
20	0.10	0.32	10,000.00	2.00	1.05	5,000.00	21.40	10.98	4,672.90	144.00	41.73	3,472.22	247.40	100.22	4,042.04
50	0.10	0.00	10,000.00	1.60	0.52	6,250.00	12.90	0.74	7,751.94	111.00	2.83	4,504.50	193.90	4.36	5,157.30
100	0.10	0.32	10,000.00	1.20	0.42	8,333.33	10.50	0.53	9,523.81	66.30	1.34	7,541.48	148.30	3.33	6,743.09
200	0.10	0.32	10,000.00	1.00	0.00	10,000.00	57.20	16.24	1,748.25	67.80	6.09	7,374.63	194.30	4.72	5,146.68
500	0.10	0.00	10,000.00	1.50	0.97	6,666.67	11.10	0.74	9,009.01	66.40	0.70	7,530.12	194.60	0.84	5,138.75
1,000	0.30	0.48	3,333.33	1.00	0.00	10,000.00	10.60	0.52	9,433.96	67.60	2.99	7,396.45	150.40	0.70	6,648.94
2,000	0.10	0.32	10,000.00	1.30	0.67	7,692.31	10.80	0.42	9,259.26	124.10	18.32	4,029.01	203.80	18.61	4,906.77
5,000	0.20	0.42	5,000.00	1.00	0.00	10,000.00	11.00	0.82	9,090.91	68.80	0.63	7,267.44	162.90	23.71	6,138.74
10,000	0.10	0.32	10,000.00	1.00	0.00	10,000.00	10.60	0.52	9,433.96	82.20	41.39	6,082.73	233.10	54.13	4,290.00
20,000	0.20	0.42	5,000.00	1.30	0.48	7,692.31	11.30	0.48	8,849.56	69.10	0.57	7,235.89	174.40	55.74	5,733.94
50,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
100,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
			7,878.79			7,696.81			7,652.63			6,080.98			5,368.03

Table A.24: Detailed results of removing data from RDF data replicas using the Androjena framework on the Samsung Galaxy S I9000

Model Size	Remove 1	Std. Dev.	Triples/Sec	Remove 10	Std. Dev.	Triples/Sec	Remove 100	Std. Dev.	Triples/Sec	Remove 500	Std. Dev.	Triples/Sec	Remove 1000	Std. Dev.	Triples/Sec
10	0.90	0.88	1,111.11	8.40	10.57	1,190.48	82.60	62.80	1,210.65	532.60	148.72	938.79	712.10	13.84	1,404.30
20	0.80	0.42	1,250.00	5.40	1.35	1,851.85	40.80	1.03	2,450.98	466.10	145.79	1,072.73	771.30	227.68	1,296.51
50	0.50	0.53	2,000.00	4.50	1.08	2,222.22	49.80	13.26	2,008.03	627.40	137.81	796.94	729.50	23.77	1,370.80
100	0.50	0.53	2,000.00	6.20	2.04	1,612.90	54.10	19.03	1,848.43	279.70	83.99	1,787.63	586.40	129.45	1,705.32
200	0.80	0.63	1,250.00	4.70	1.16	2,127.66	43.70	16.26	2,288.33	238.90	7.05	2,092.93	760.50	183.31	1,314.92
500	0.50	0.53	2,000.00	9.60	3.75	1,041.67	42.20	0.79	2,369.67	245.20	12.49	2,039.15	950.20	252.99	1,052.41
1,000	0.60	0.52	1,666.67	6.00	1.89	1,666.67	48.00	13.28	2,083.33	259.00	51.78	1,930.50	650.50	219.45	1,537.28
2,000	0.20	0.42	5,000.00	4.60	2.12	2,173.91	45.10	30.36	2,217.29	487.20	317.39	1,026.27	957.00	595.88	1,044.93
5,000	0.50	0.53	2,000.00	3.70	2.06	2,702.70	35.10	18.87	2,849.00	207.10	98.96	2,414.29	518.40	271.57	1,929.01
10,000	0.30	0.48	3,333.33	4.50	3.44	2,222.22	32.90	18.48	3,039.51	309.40	224.37	1,616.03	489.30	293.13	2,043.74
20,000	0.30	0.48	3,333.33	2.20	1.81	4,545.45	20.70	13.52	4,830.92	143.30	91.26	3,489.18	345.10	309.22	2,897.71
50,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
100,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
			2,267.68			2,123.43			2,472.38			1,745.86			1,599.72

TABLE A.25: Detailed results of removing data from RDF data replicas using the μJena framework on the HTC G1

Model Size	Remove 1	Std. Dev.	Triples/Sec	Remove 10	Std. Dev.	Triples/Sec	Remove 100	Std. Dev.	Triples/Sec	Remove 500	Std. Dev.	Triples/Sec	Remove 1000	Std. Dev.	Triples/Sec
10	2.40	0.52	416.67	46.60	1.07	214.59	2,741.70	139.18	36.47	62,059.00	1,958.61	8.06	256,892.60	7,458.10	3.89
20	2.50	0.53	400.00	48.20	0.92	207.47	2,731.80	71.63	36.61	60,932.50	1,348.92	8.21	255,550.90	6,289.46	3.91
50	2.90	0.32	344.83	52.60	0.84	190.11	2,753.80	78.10	36.31	61,616.50	968.87	8.11	254,681.60	4,402.37	3.93
100	3.40	0.52	294.12	55.80	0.63	179.21	2,856.60	68.29	35.01	60,261.60	1,089.89	8.30	249,372.30	3,750.86	4.01
200	9.50	0.53	105.26	120.20	4.71	83.19	3,451.60	120.84	28.97	63,816.60	1,321.98	7.83	260,120.90	6,811.07	3.84
500	13.60	0.70	73.53	165.70	10.70	60.35	4,052.50	168.52	24.68	69,327.20	2,845.36	7.21	273,877.90	10,952.31	3.65
1,000	57.90	17.04	17.27	499.60	21.69	20.02	7,594.10	235.52	13.17	87,920.40	2,854.93	5.69	319,009.40	12,655.65	3.13
2,000	188.40	9.69	5.31	2,014.90	150.04	4.96	22,448.00	175.10	4.45	161,709.90	2,222.77	3.09	458,630.10	5,961.29	2.18
5,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
10,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
20,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
50,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
100,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
			207.12			119.99			26.96			7.06			3.57

TABLE A.26: Detailed results of removing data from RDF data replicas using the μJena framework on the Motorola Milestone

Model Size	Remove 1	Std. Dev.	Triples/Sec	Remove 10	Std. Dev.	Triples/Sec	Remove 100	Std. Dev.	Triples/Sec	Remove 500	Std. Dev.	Triples/Sec	Remove 1000	Std. Dev.	Triples/Sec
10	1.00	0.00	1,000.00	17.00	0.47	588.24	1,019.10	59.76	98.13	22,589.50	374.10	22.13	96,280.90	680.77	10.39
20	1.10	0.32	909.09	17.30	0.48	578.03	1,013.30	66.17	98.69	22,791.90	360.91	21.94	97,901.80	1,467.75	10.21
50	1.10	0.32	909.09	18.70	0.95	534.76	1,001.00	59.08	99.90	22,733.40	239.12	21.99	96,672.10	275.04	10.34
100	1.20	0.42	833.33	20.70	1.64	483.09	1,036.30	97.18	96.50	22,752.00	171.16	21.98	96,897.50	395.54	10.32
200	4.40	0.52	227.27	51.60	1.90	193.80	1,317.60	79.94	75.90	34,251.40	22,896.56	14.60	185,293.30	34,235.96	5.40
500	5.70	0.48	175.44	63.80	1.03	156.74	2,583.40	3,700.09	38.71	46,330.80	25,943.65	10.79	186,795.30	26,704.57	5.35
1,000	8.70	3.30	114.94	85.50	1.84	116.96	4,399.40	8,611.92	22.73	46,239.30	27,692.03	10.81	171,070.30	46,057.72	5.85
2,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
5,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
10,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
20,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
50,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
100,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
			595.60			378.80			75.79			17.75			8.27

TABLE A.27: Detailed results of removing data from RDF data replicas using the μJena framework on the Samsung Galaxy S I9000

Model Size	Remove 1	Std. Dev.	Triples/Sec	Remove 10	Std. Dev.	Triples/Sec	Remove 100	Std. Dev.	Triples/Sec	Remove 500	Std. Dev.	Triples/Sec	Remove 1000	Std. Dev.	Triples/Sec
10	0.20	0.42	5,000.00	6.50	7.28	1,538.46	223.70	16.64	447.03	5,469.20	295.87	91.42	23,489.10	514.62	42.57
20	0.10	0.00	10,000.00	5.00	1.89	2,000.00	220.00	9.49	454.55	5,580.30	217.24	89.60	24,518.20	381.19	40.79
50	0.30	0.48	3,333.33	5.20	1.93	1,923.08	229.70	22.09	435.35	5,565.00	236.67	89.85	24,537.20	348.20	40.75
100	0.30	0.48	3,333.33	4.50	0.53	2,222.22	260.40	34.67	384.02	5,347.30	304.32	93.51	23,678.50	1,301.34	42.23
200	1.00	0.82	1,000.00	16.70	18.38	598.80	308.50	36.42	324.15	5,971.30	133.64	83.73	25,253.80	796.74	39.60
500	1.10	0.32	909.09	13.40	1.90	746.27	310.40	48.45	322.16	5,738.00	324.58	87.14	24,284.50	1,646.47	41.18
1,000	1.40	0.70	714.29	16.20	1.14	617.28	327.70	20.36	305.16	5,691.30	163.46	87.85	23,290.20	313.61	42.94
2,000	20.80	3.26	48.08	189.20	1.93	52.85	2,127.60	53.61	47.00	14,911.50	117.59	33.53	41,556.40	548.38	24.06
5,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
10,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
20,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
50,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
100,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
			3,042.27			1,212.37			339.93			82.08			39.27

TABLE A.28: Detailed results of removing data from RDF data replicas using the μJena framework on the Dell Streak 5

Model Size	Remove 1	Std. Dev.	Triples/Sec	Remove 10	Std. Dev.	Triples/Sec	Remove 100	Std. Dev.	Triples/Sec	Remove 500	Std. Dev.	Triples/Sec	Remove 1000	Std. Dev.	Triples/Sec
10	0.40	0.52	2,500.00	5.00	2.54	2,000.00	234.70	118.53	426.08	5,255.60	138.69	95.14	22,326.50	383.15	44.79
20	0.50	0.53	2,000.00	4.30	0.67	2,325.58	202.70	6.06	493.34	5,283.30	43.75	94.64	22,394.10	110.33	44.65
50	0.40	0.52	2,500.00	4.40	0.52	2,272.73	208.60	12.31	479.39	5,320.70	33.52	93.97	22,387.90	112.86	44.67
100	0.70	0.48	1,428.57	5.00	0.67	2,000.00	242.00	9.90	413.22	5,318.30	44.08	94.02	22,513.10	65.04	44.42
200	0.60	0.52	1,666.67	6.40	0.52	1,562.50	224.00	7.36	446.43	5,461.80	46.58	91.54	22,657.90	130.61	44.13
500	8.90	17.46	112.36	9.10	0.74	1,098.90	251.50	4.12	397.61	5,589.20	105.24	89.46	22,964.80	213.22	43.54
1,000	1.50	0.53	666.67	15.00	0.67	666.67	307.20	3.43	325.52	5,820.90	89.18	85.90	23,532.80	199.71	42.49
2,000	26.10	0.74	38.31	251.30	6.34	39.79	2,744.80	39.61	36.43	17,768.30	304.74	28.14	47,548.50	328.91	21.03
5,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
10,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
20,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
50,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
100,000	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
			1,364.07			1,495.77			377.25			84.10			41.22

TABLE A.29: Detailed results of retrieving elements from RDF data replicas using the Androjena, μJena, and Mobile RDF frameworks

		Model Size (Triples)	10	20	50	100	200	500	1,000	2,000	5,000	10,000	20,000	50,000	100,000
Androjena	HTC G1	Execution Time (ms)	3.90	3.50	3.20	3.30	3.70	3.40	3.20	4.00	3.60	3.10	DNF	DNF	DNF
		Standard Deviation	1.85	0.71	0.42	0.48	0.48	0.52	0.42	2.16	0.52	0.32	DNF	DNF	DNF
	Motorola Milestone	Execution Time (ms)	1.00	0.90	1.10	0.90	1.10	0.90	0.90	1.00	0.90	0.90	1.10	1.20	DNF
		Standard Deviation	0.47	0.32	0.32	0.32	0.32	0.32	0.32	0.82	0.32	0.32	0.32	0.63	DNF
	Samsung Galaxy	Execution Time (ms)	0.90	0.50	0.60	0.50	0.50	0.60	0.70	1.10	0.50	0.60	0.90	0.90	DNF
		Standard Deviation	0.57	0.53	0.52	0.53	0.53	0.52	0.48	0.99	0.53	0.52	0.99	0.88	DNF
	Dell Streak	Execution Time (ms)	2.20	0.60	0.90	1.00	0.70	0.60	0.60	1.20	0.90	1.30	1.20	2.80	DNF
		Standard Deviation	1.14	0.52	0.32	0.00	0.48	0.70	0.70	0.92	0.32	0.82	0.42	1.23	DNF

		Model Size (Triples)	10	20	50	100	200	500	1,000	2,000	5,000	10,000	20,000	50,000	100,000
µJena	HTC G1	Execution Time (ms)	149.60	159.10	183.20	154.10	184.00	193.20	203.40	391.20	DNF	DNF	DNF	DNF	DNF
		Standard Deviation	52.82	4.58	51.48	2.18	10.88	2.90	19.36	69.51	DNF	DNF	DNF	DNF	DNF
	Motorola Milestone	Execution Time (ms)	267.10	284.60	5,152.10	268.40	3,727.40	290.80	324.50	DNF	DNF	DNF	DNF	DNF	DNF
		Standard Deviation	1.37	47.01	15,447.69	2.99	10,932.56	44.70	42.52	DNF	DNF	DNF	DNF	DNF	DNF
	Samsung Galaxy	Execution Time (ms)	13.40	12.40	12.30	12.30	13.00	13.30	14.80	17.20	21.90	DNF	DNF	DNF	DNF
		Standard Deviation	2.46	0.52	0.48	0.48	0.67	0.48	2.53	2.78	0.32	DNF	DNF	DNF	DNF
	Dell Streak	Execution Time (ms)	16.40	16.70	16.70	17.30	17.50	18.00	19.10	46.70	53.00	DNF	DNF	DNF	DNF
		Standard Deviation	0.52	0.48	0.48	1.89	0.71	0.47	0.74	7.15	7.07	DNF	DNF	DNF	DNF

		Model Size (Triples)	10	20	50	100	200	500	1,000	2,000	5,000	10,000	20,000	50,000	100,000
Mobile RDF	HTC G1	Execution Time (ms)	0.50	0.90	1.00	1.90	5.20	8.80	17.50	33.00	80.20	108.50	DNF	DNF	DNF
		Standard Deviation	0.53	0.74	0.00	0.88	6.61	0.63	1.08	1.94	3.99	20.21	DNF	DNF	DNF
	Motorola Milestone	Execution Time (ms)	0.20	0.40	0.10	0.70	1.40	3.20	6.70	14.20	37.30	86.80	92.00	DNF	DNF
		Standard Deviation	0.42	0.52	0.32	0.48	0.52	0.42	0.48	0.79	3.71	32.18	16.54	DNF	DNF
	Samsung Galaxy	Execution Time (ms)	0.30	0.50	0.30	0.20	0.20	1.00	2.00	4.10	10.40	19.40	40.70	DNF	DNF
		Standard Deviation	0.48	0.53	0.48	0.42	0.42	0.00	0.00	1.45	1.51	1.96	1.25	DNF	DNF
	Dell Streak	Execution Time (ms)	0.60	0.40	0.20	0.50	0.60	1.10	1.90	4.30	10.10	21.00	45.00	DNF	DNF
		Standard Deviation	0.97	0.52	0.42	0.53	0.52	0.32	0.32	1.70	0.32	0.00	0.67	DNF	DNF

TABLE A.30: Detailed results of constructing in-memory RDF graphs using the Androjena, μJena, and Mobile RDF frameworks

		Model Size (Triples)	10	20	50	100	200	500	1,000	2,000	5,000	10,000	20,000	50,000	100,000
Androjena	HTC G1	Execution Time (ms)	20.50	31.80	77.80	150.00	302.30	823.00	1,811.40	3,819.00	10,069.70	20,343.00	41,627.30	DNF	DNF
		Standard Deviation	6.40	1.40	2.04	3.46	3.47	63.86	12.47	104.14	78.39	159.39	303.21	DNF	DNF
		Triples per Second	487.80	628.93	642.67	666.67	661.59	607.53	552.06	523.70	496.54	491.57	480.45	DNF	DNF
	Motorola Milestone	Execution Time (ms)	8.60	18.70	45.90	76.00	112.90	251.80	578.80	1,170.60	2,973.10	6,144.30	12,494.20	DNF	DNF
		Standard Deviation	0.70	3.02	3.03	7.04	2.85	6.97	7.27	21.36	41.75	23.45	74.18	DNF	DNF
		Triples per Second	1,162.79	1,069.52	1,089.32	1,315.79	1,771.48	1,985.70	1,727.71	1,708.53	1,681.75	1,627.52	1,600.74	DNF	DNF
	Samsung Galaxy	Execution Time (ms)	18.20	38.90	97.70	154.80	241.20	364.10	563.70	854.40	1,718.10	3,270.30	6,498.80	18,913.70	DNF
		Standard Deviation	6.25	6.31	17.41	34.78	15.25	4.07	15.11	28.18	83.38	59.84	38.06	713.87	DNF
		Triples per Second	549.45	514.14	511.77	645.99	829.19	1,373.25	1,773.99	2,340.82	2,910.19	3,057.82	3,077.49	2,643.59	DNF

		Model Size (Triples)	10	20	50	100	200	500	1,000	2,000	5,000	10,000	20,000	50,000	100,000
µJena	HTC G1	Execution Time (ms)	55.30	147.00	553.30	2,138.90	9,914.40	67,168.30	DNF	DNF	DNF	DNF	DNF	DNF	DNF
		Standard Deviation	3.27	10.74	23.38	75.48	308.93	5,054.04	DNF	DNF	DNF	DNF	DNF	DNF	DNF
		Triples per Second	180.83	136.05	90.37	46.75	20.17	7.44	DNF	DNF	DNF	DNF	DNF	DNF	DNF
	Motorola Milestone	Execution Time (ms)	33.90	84.50	242.50	858.50	4,107.80	29,055.20	143,648.50	DNF	DNF	DNF	DNF	DNF	DNF
		Standard Deviation	8.50	5.25	15.47	22.96	116.82	2,772.42	12,689.76	DNF	DNF	DNF	DNF	DNF	DNF
		Triples per Second	294.99	236.69	206.19	116.48	48.69	17.21	6.96	DNF	DNF	DNF	DNF	DNF	DNF
	Samsung Galaxy	Execution Time (ms)	55.30	132.30	283.40	630.00	2,345.50	14,718.90	61,784.50	236,003.38	DNF	DNF	DNF	DNF	DNF
		Standard Deviation	26.25	16.59	20.13	36.28	101.60	1,261.50	14,590.10	63,201.29	DNF	DNF	DNF	DNF	DNF
		Triples per Second	180.83	151.17	176.43	158.73	85.27	33.97	16.19	8.47	DNF	DNF	DNF	DNF	DNF

		Model Size (Triples)	10	20	50	100	200	500	1,000	2,000	5,000	10,000	20,000	50,000	100,000
Mobile RDF	HTC G1	Execution Time (ms)	13.80	21.10	47.30	92.60	181.40	603.70	1,222.90	2,644.20	6,318.20	12,515.40	DNF	DNF	DNF
		Standard Deviation	7.47	0.74	1.83	2.12	2.88	4.76	7.89	59.62	73.28	174.13	DNF	DNF	DNF
		Triples per Second	724.64	947.87	1,057.08	1,079.91	1,102.54	828.23	817.73	756.37	791.36	799.02	DNF	DNF	DNF
	Motorola Milestone	Execution Time (ms)	5.30	12.20	30.20	53.30	87.30	234.50	456.90	921.40	2,154.70	4,332.60	9,383.30	DNF	DNF
		Standard Deviation	1.57	1.14	4.89	2.21	6.02	6.22	10.05	14.79	11.98	81.84	110.98	DNF	DNF
		Triples per Second	1,886.79	1,639.34	1,655.63	1,876.17	2,290.95	2,132.20	2,188.66	2,170.61	2,320.51	2,308.08	2,131.45	DNF	DNF
	Samsung Galaxy	Execution Time (ms)	11.10	24.90	64.80	114.90	189.90	362.40	461.90	678.50	1,320.90	2,358.10	4,824.40	15,404.60	DNF
		Standard Deviation	3.96	4.23	3.29	7.49	24.04	22.70	33.52	18.34	90.27	39.50	51.57	667.82	DNF
		Triples per Second	900.90	803.21	771.60	870.32	1,053.19	1,379.69	2,164.97	2,947.68	3,785.30	4,240.70	4,145.59	3,245.78	DNF

Bibliography

[ABK+07] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In *In 6th Int'l Semantic Web Conference, Busan, Korea*, pages 11–15. Springer, 2007.

[Abo99] Gregory D. Abowd. Software engineering issues for ubiquitous computing. In *ICSE '99: Proceedings of the 21st international conference on Software engineering*, pages 75–84, New York, NY, USA, 1999. ACM.

[ACC09] Martin Azizyan, Ionut Constandache, and Romit Roy Choudhury. Surroundsense: mobile phone localization via ambience fingerprinting. In Kang G. Shin, Yongguang Zhang, Rajive Bagrodia, and Ramesh Govindan, editors, *MOBICOM*, pages 261–272. ACM, 2009.

[ADB+99] Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, pages 304–307, London, UK, 1999. Springer-Verlag.

[AH08] Dean Allemang and James A. Hendler. *Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL*. Morgan Kaufmann, Burlington, MA, 2008.

[AMMH09] Daniel J. Abadi, Adam Marcus, Samuel R. Madden, and Kate Hollenbach. Sw-store: a vertically partitioned dbms for semantic web data management. *The VLDB Journal*, 18:385–406, April 2009.

[AMR02] Gregory D. Abowd, Elizabeth D. Mynatt, and Tom Rodden. The human experience. *IEEE Pervasive Computing*, 1:48–57, 2002.

[And10] Android developer's guide, 12 2010.

[ANH07] C. B. Anagnostopoulos, Y. Ntarladimas, and Stathes Hadjiefthymiades. Situational computing: An innovative architecture with imprecise reasoning. *Journal of Systems and Software*, 80(12):1993–2014, 2007.

[ATH07] Christos B. Anagnostopoulos, Athanasios Tsounis, and Stathes Hadjiefthymiades. Context awareness in mobile computing environments. *Wirel. Pers. Commun.*, 42(3):445–464, 2007.

[Bar03] Louise Barkhuus. Context information in mobile telephony. In Luca Chittaro, editor, *Mobile HCI*, volume 2795 of *Lecture Notes in Computer Science*, pages 451–455. Springer, 2003.

[BB08] Christian Becker and Christian Bizer. Dbpedia mobile: A location-enabled linked data browser. In *Linked Data on the Web (LDOW2008)*, 2008.

[BB09a] Christian Becker and Chris Bizer. Marbles, 2009.

[BB09b] Christian Becker and Christian Bizer. Exploring the geospatial semantic web with dbpedia mobile. *J. Web Sem.*, 7(4):278–286, 2009.

[BBL08] David Beckett and Tim Berners-Lee. Turtle – terse rdf triple language. W3c team submission, W3C, January 2008.

[BC04] G. Biegel and V. Cahill. A framework for developing mobile, context-aware applications. pages 361–365, March 2004.

[BCH07] Chris Bizer, Richard Cyganiak, and Tom Heath. *How to publish Linked Data on the Web*, 2007.

[BCM+03] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[BCQ+07] Cristiana Bolchini, Carlo A. Curino, Elisa Quintarelli, Fabio A. Schreiber, and Letizia Tanca. A data-oriented survey of context models. *SIGMOD Rec.*, 36(4):19–26, 2007.

[BCR04] Margaret Burnett, Curtis Cook, and Gregg Rothermel. End-user software engineering. *Commun. ACM*, 47:53–58, September 2004.

[BD05a] Christian Becker and Frank Duerr. On location models for ubiquitous computing. *Personal Ubiquitous Comput.*, 9(1):20–31, 2005.

[BD05b] Nicholas A. Bradley and Mark D. Dunlop. Toward a multidisciplinary model of context to support context-aware computing. *Hum.-Comput. Interact.*, 20(4):403–446, 2005.

[BDR07] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. A survey on context-aware systems. *Int. J. Ad Hoc Ubiquitous Comput.*, 2(4):263–277, 2007.

[Bec04] Dave Beckett. *RDF/XML Syntax Specification (Revised)*. World Wide Web Consortium, 2004.

[BEK+00] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, and Dave Winer. Simple object access protocol (SOAP) 1.1. W3C Note NOTE-SOAP-20000508, World Wide Web Consortium, May 2000.

[BG03] Michael Beigl and Hans Gellersen. Smart-its: An embedded platform for smart objects. In *In Proc. Smart Objects Conference (SOC 2003*, pages 15–17, 2003.

[BG04] Dan Brickley and R.V. Guha. *RDF Vocabulary Description Language 1.0: RDF Schema (W3C Recommendation 10 Februar 2004)*. World Wide Web Consortium, 2004.

[BGSA05] Abdelkrim Beloued, Jean-Marie Gilliot, Maria-Teresa Segarra, and Françoise André. Dynamic Data Replication and Consistency in Mobile Environments. In *Proc. of the 2nd international doctoral symposium on Middleware*, pages 1–5, New York, NY, USA, 2005. ACM.

[Bha99] Bharat Bhargava. Concurrency control in database systems. *IEEE Trans. on Knowl. and Data Eng.*, 11:3–16, January 1999.

[BHBL08] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked Data: Principles and State of the Art. In *17th World Wide Web Conference (WWW2008), W3C track*, April 2008.

[BHBL09] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data - the story so far. *International Journal on Semantic Web and Information Systems*, 5(3):1–22, 2009.

[BHG87] Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[BHK+10] Harold Boley, Gary Hallmark, Michael Kifer, Adrian Paschke, Axel Polleres, and Dave Reynolds. *RIF Core Dialect (W3C Recommendation 22 June 2010)*. World Wide Web Consortium, June 2010. Available at `http://www.w3.org/TR/rif-core/`.

[Biz09] Christian Bizer. The emerging web of linked data. *IEEE Intelligent Systems*, 24:87–92, 2009.

[BJRGN10] Francisco Borrego-Jaraba, Irene Luque Ruiz, and Miguel Ángel Gómez-Nieto. Nfc solution for the development of smart scenarios supporting tourism applications and surfing in urban environments. In *Proceedings of the 23rd international conference on Industrial engineering and other applications of applied intelligent systems - Volume Part III*, IEA/AIE'10, pages 229–238, Berlin, Heidelberg, 2010. Springer-Verlag.

[BKL+08a] Sebastian Boehm, Johan Koolwaaij, Marko Luther, Bertrand Souville, Matthias Wagner, and Martin Wibbels. Introducing iyouit. *The Semantic Web - ISWC 2008*, pages 804–817, 2008///.

[BKL08b] Sebastian Böhm, Johan Koolwaaij, and Marko Luther. Share whatever you like. *ECEASST*, 11, 2008.

[BKL+08c] Sebastian Böhm, Johan Koolwaaij, Marko Luther, Bertrand Souville, Matthias Wagner, and Martin Wibbels. Iyouit - share, life, blog, play. In Christian Bizer and Anupam Joshi, editors, *International Semantic Web Conference (Posters Demos)*, volume 401 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[BKvH02] Jeen Broekstra, Arjohn Kampman, and Frank van Harmelen. Sesame: A generic architecture for storing and querying RDF and RDF schema. In I. Horrocks and J. Hendler, editors, *Proceedings of the First Internation Semantic Web Conference*, number 2342 in Lecture Notes in Computer Science, pages 54–68. Springer Verlag, July 2002.

[BL97] Tim Berners-Lee. Uri references: Fragment identifiers on uris - axioms of web architecture. Technical report, April 1997.

[BL98] Tim Berners-Lee. Why rdf model is different from the xml model, September 1998.

[BL02] Tim Berners-Lee. What do http uris identify?, 07 2002.

[BL05] Tim Berners-Lee. What http uris identify, 06 2005.

[BL06a] Tim Berners-Lee. *Linked Data – Design Issues*, 2006.

[BL06b] Tim Berners-Lee. Notation 3, March 2006.

[BLC08] Tim Berners-Lee and Dan Connolly. Notation3 (n3): A readable rdf syntax. W3c team submission, W3C, January 2008.

[BLFM05] T. Berners-Lee, R. Fielding, and L. Masinter. *Uniform Resource Identifier (URI): Generic Syntax (RFC 3986)*. Network Working Group, January 2005.

[BLHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 284(5):34–43, 2001.

[BLMM94] T. Berners-Lee, L. Masinter, and M. McCahill. Uniform Resource Locators (URL). RFC 1738 (Proposed Standard), December 1994. Obsoleted by RFCs 4248, 4266, updated by RFCs 1808, 2368, 2396, 3986.

[BM04] Paul V. Biron and Ashok Malhotra. XML Schema Part 2: Datatypes Second Edition, W3C Recommendation. October 2004.

[BM07] Dan Brickley and Libby Miller. Foaf vocabulary specification 0.91, 2007.

[Bon04] Elena Paslaru Bontas. Representing context on the semantic web. *Doktorandenworkshop Technologien und Anwendungen von XML, Berliner XML-Tage*, 2004.

[BPM+08] Diego Berrueta, Jon Phipps, Alistair Miles, Thomas Baker, and Ralph Swick. Best practice recipes for publishing rdf vocabularies, August 2008.

[Bro96] P. J. Brown. The stick-e document: a framework for creating context-aware applications. In *Proceedings of EP'96, Palo Alto*, pages 259–272. also published in it EP–odd, January 1996.

[BS03] Thomas Buchholz and Michael Schiffers. Quality of context: What it is and why we need it. In *In Proceedings of the 10th Workshop of the OpenView University Association: OVUA'03*, 2003.

[BS04] Richard Boardman and M. Angela Sasse. "stuff goes into the computer and doesn't come out": a cross-tool study of personal information management. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '04, pages 583–590, New York, NY, USA, 2004. ACM.

[BTC06] Yingyi Bu, Xianping Tao, and Shaxun Chen. Managing quality of context in pervasive computing. In *In: QSIC 2006: Proceedings of the Sixth International Conference on Quality Software*, pages 193–200, 2006.

[BZD02] Michael Beigl, Tobias Zimmer, and Christian Decker. A location model for communicating and processing of context. *Personal and Ubiquitous Computing*, 6(5/6):341–357, 2002.

[CBHS05] Jeremy J. Carroll, Christian Bizer, Pat Hayes, and Patrick Stickler. Named graphs. *Journal of Web Semantics*, 3(4):247–267, 2005.

[CCDG05] Joëlle Coutaz, James L. Crowley, Simon Dobson, and David Garlan. Context is key. *Communications of the ACM - SPECIAL ISSUE: The disappearing computer*, 48(3):49–53, 2005.

[CCMS10] Alessandro Ciaramella, Mario G. C. A. Cimino, Francesco Marcelloni, and Umberto Straccia. Combining fuzzy logic and semantic web to enable situation-awareness in service recommendation. In *Proceedings of the 21st international conference on Database and expert systems applications: Part I*, DEXA'10, pages 31–45, Berlin, Heidelberg, 2010. Springer-Verlag.

[CCSC07] Andrew Carton, Siobhan Clarke, Aline Senart, and Vinny Cahill. Aspect-oriented model-driven development for mobile context-aware computing. In *SEPCASE '07: Proceedings of the 1st International Workshop on Software Engineering for Pervasive Computing Applications, Systems, and Environments*, page 5, Washington, DC, USA, 2007. IEEE Computer Society.

[CDA07] Abdelghani Chibani, Karim Djouani, and Yacine Amirat. Semantic middleware for context services composition in ubiquitous computing. In *MOBILWARE '08: Proceedings of the 1st international conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications*, pages 1–6, ICST, Brussels, Belgium, Belgium, 2007. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[CDY90] B. Ciciani, D. M. Dias, and P. S. Yu. Analysis of replication in distributed database systems. *IEEE Trans. on Knowl. and Data Eng.*, 2(2):247–261, 1990.

[CEM03] Licia Capra, Wolfgang Emmerich, and Cecilia Mascolo. Carisma: Context-aware reflective middleware system for mobile applications. *IEEE Transactions on Software Engineering*, 29:929–945, 2003.

[CFJ03] Harry Chen, Tim Finin, and Anupam Joshi. An ontology for context-aware pervasive computing environments. *Knowl. Eng. Rev.*, 18(3):197–207, 2003.

[Che04] Harry Chen. *An Intelligent Broker Architecture for Pervasive Context-Aware Systems*. PhD thesis, University of Maryland, Baltimore County, December 2004.

[CJ04] Harry Chen and Anupam Joshi. An Ontology for Context-Aware Pervasive Computing Environments. *Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review*, 18(3):197–207, May 2004.

[CK00] Guanling Chen and David Kotz. A survey of context-aware mobile computing research. Technical report, Hanover, NH, USA, 2000.

[CK02] Guanling Chen and David Kotz. Context aggregation and dissemination in ubiquitous computing systems. In *Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications*, WMCSA '02, pages 105–, Washington, DC, USA, 2002. IEEE Computer Society.

[CPFJ04] Harry Chen, Filip Perich, Tim Finin, and Anupam Joshi. Soupa: Standard ontology for ubiquitous and pervasive applications. In *In International Conference on Mobile and Ubiquitous Systems: Networking and Services*, pages 258–267, 2004.

[CQ09] Gong Cheng and Yuzhong Qu. Searching linked objects with falcons: Approach, implementation and evaluation. *International Journal on Semantic Web and Information Systems*, 5(3):49–70, 2009.

[CR90] Panayiotis K. Chrysanthis and Krithi Ramamritham. Acta: a framework for specifying and reasoning about transaction structure and behavior. In *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, SIGMOD '90, pages 194–203, New York, NY, USA, 1990. ACM.

[CR91a] Panayiotis K. Chrysanthis and Krithi Ramamritham. A unifying framework for transactions in competitive and cooperative environments. *Off. Knowl. Eng.*, 4:3–21, May 1991.

[CR91b] Panos K. Chrysanthis and Krithi Ramamritham. Acta: The saga continues. Technical report, Amherst, MA, USA, 1991.

[CR91c] Panos K. Chrysanthis and Krithi Ramamritham. A formalism for extended transaction model. In *Proceedings of the 17th International Conference on Very Large Data Bases*, VLDB '91, pages 103–112, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc.

[CR94] Panos K. Chrysanthis and Krithi Ramamritham. Synthesis of extended transaction models using acta. *ACM Trans. Database Syst.*, 19:450–491, September 1994.

[CRL+09] Alejandro Cadenas, Carlos Ruiz, Iker Larizgoitia, Raúl García-Castro, Carlos Lamsfus, Iñaki Vázquez, Marta González, David Martín, and María Poveda. Context management in mobile environments: a semantic approach. In *Proceedings of the 1st Workshop on Context, Information and Ontologies*, CIAO '09, pages 2:1–2:8, New York, NY, USA, 2009. ACM.

[CS08] Karen Church and Barry Smyth. Understanding mobile information needs. In *MobileHCI '08: Proceedings of the 10th international conference on Human computer interaction with mobile devices and services*, pages 493–494, New York, NY, USA, 2008. ACM.

[CSC04] Isabel F. Cruz, William Sunna, and Anjli Chaudhry. Ontology alignment for real-world applications. In *Proceedings of the 2004 annual national conference on Digital government research*, dg.o '04, pages 96:1–96:2. Digital Government Society of North America, 2004.

[CSH06] Namyoun Choi, Il-Yeol Song, and Hyoil Han. A survey on ontology mapping. *SIGMOD Rec.*, 35:34–41, September 2006.

[CX06] Guoray Cai and Yinkun Xue. Activity-oriented context-aware adaptation assisting mobile geo-spatial activities. In *IUI '06: Proceedings of the 11th international conference on Intelligent user interfaces*, pages 354–356, New York, NY, USA, 2006. ACM.

[DAPW97] Anind K. Dey, Gregory D. Abowd, Mike Pinkerton, and Andrew Wood. Cyberdesk: A framework for providing self-integrating ubiquitous software services. In *ACM Symposium on User Interface Software and Technology*, pages 75–76, 1997.

[DAS01] Anind K. Dey, Gregory D. Abowd, and Daniel Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction*, 16(2):97–166, 2001.

[DE10] Jérôme David and Jérôme Euzenat. Linked data from your pocket: The android RDFContentProvider. In *9th International Semantic Web Conference (ISWC2010)*, November 2010.

[Dey98] Anind K. Dey. Context-aware computing: The cyberdesk project. In *AAAI 1998 Spring Symposium on Intelligent Environments*, pages 51–54, Palo Alto, 1998. AAAI Press.

[Dey00] Anind K. Dey. *Providing architectural support for building context-aware applications.* PhD thesis, Georgia Institute of Technology, 2000.

[Dey01] Anind K. Dey. Understanding and using context. *Personal Ubiquitous Comput.*, 5(1):4–7, 2001.

[DFP+05] Li Ding, Tim Finin, Yun Peng, Paulo Pinheiro da Silva, and Deborah L. McGuinness. Tracking rdf graph provenance using rdf molecules. In *Proceedings of the 4th International Semantic Web Conference*, November 2005.

[Dia90] Dan Diaper. *Task Analysis for Human-Computer Interaction*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1990.

[DMD+03] AnHai Doan, Jayant Madhavan, Robin Dhamankar, Pedro Domingos, and Alon Halevy. Learning to match ontologies on the semantic web. *The VLDB Journal*, 12:303–319, November 2003.

[DMDH02] AnHai Doan, Jayant Madhavan, Pedro Domingos, and Alon Halevy. Learning to map between ontologies on the semantic web. In *Proceedings of the 11th international conference on World Wide Web*, WWW '02, pages 662–673, New York, NY, USA, 2002. ACM.

[Dou04] Paul Dourish. What we talk about when we talk about context. *Personal Ubiquitous Comput.*, 8(1):19–30, 2004.

[DRD+00] Alan Dix, Tom Rodden, Nigel Davies, Jonathan Trevor, Adrian Friday, and Kevin Palfreyman. Exploiting space and location as a design framework for interactive mobile systems. *ACM Trans. Comput.-Hum. Interact.*, 7(3):285–321, 2000.

[DS06] Nick Drummond and Rob Shearer. The open world assumption. electronic, 2006.

[DWM08] Dominique Dudkowski, Harald Weinschrott, and Pedro Jose Marron. Design and implementation of a reference model for context management in mobile ad-hoc networks. In *AINAW '08: Proceedings of the 22nd International Conference on Advanced Information Networking and Applications - Workshops*, pages 832–837, Washington, DC, USA, 2008. IEEE Computer Society.

[Ege02] Max J. Egenhofer. Toward the semantic geospatial web. In Agnès Voisard and Shu-Ching Chen, editors, *ACM-GIS*, pages 1–4. ACM, 2002.

[EM07] Orri Erling and Ivan Mikhailov. RDF support in the virtuoso DBMS. In Sören Auer, Christian Bizer, Claudia Müller, and Anna V. Zhdanova, editors, *CSSW*, volume 113 of *LNI*, pages 59–68. GI, 2007.

[EPR08] Jérôme Euzenat, Jérôme Pierson, and Fano Ramparany. Dynamic context management for pervasive applications. *The Knowledge Engineering Review*, 23(1):21–49, 2008.

[Eri02] Thomas Erickson. Some problems with the notion of context-aware computing. *Commun. ACM*, 45(2):102–104, 2002.

[Euz05] Jérôme Euzenat. Alignment infrastructure for ontology mediation and other applications. In Martin Hepp, Axel Polleres, Frank van Harmelen, and Michael R. Genesereth, editors, *MEDIATE2005*, volume 168 of *CEUR Workshop Proceedings*, pages 81–95. CEUR-WS.org, 2005. online http://CEUR-WS.org/Vol-168/MEDIATE2005-paper6.pdf.

[FAS09] Thomas Franz, Scherp Ansgar, and Steffen Staab. Are semantic desktops better?: summative evaluation comparing a semantic against a conventional desktop. In *Proceedings of the fifth international conference on Knowledge capture*, K-CAP '09, pages 1–8, New York, NY, USA, 2009. ACM.

[FGM+99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1 (RFC 2616), 1999.

[FL04] Peter A. Flach and Nicolas Lachiche. Naive bayesian classification of structured data. *Machine Learning*, 57:233–269, 2004. 10.1023/B:MACH.0000039778.69032.ab.

[FMGI06] Damien Fournier, Sonia Ben Mokhtar, Nikolaos Georgantas, and Valérie Issarny. Towards ad hoc contextual services for pervasive computing. In *MW4SOC '06: Proceedings of the 1st workshop on Middleware for Service Oriented Computing (MW4SOC 2006)*, pages 36–41, New York, NY, USA, 2006. ACM.

[FZ94] George H. Forman and John Zahorjan. The challenges of mobile computing. *Computer*, 27(4):38–47, 1994.

[GASW07] Guido Gehlen, Fahad Aijaz, Muhammad Sajjad, and Bernhard Walke. A mobile context dissemination middleware. In *Proceedings of the International Conference on Information Technology*, ITNG '07, pages 155–160, Washington, DC, USA, 2007. IEEE Computer Society.

[GB04] Jan Grant and Dave Beckett. *RDF Test Cases*. World Wide Web Consortium, February 2004.

[Geh08] Jan D. Gehrke. Evaluating situation awareness of autonomous systems. In *PerMIS '08: Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems*, pages 206–213, New York, NY, USA, 2008. ACM.

[GK96] Barbara J. Grosz and Sarit Kraus. Collaborative plans for complex group action. *Artif. Intell.*, 86(2):269–357, 1996.

[GM03] R. Guha and John McCarthy. Varieties of contexts. In *Proceedings of CONTEXT 2003*, volume 2680 / 2003, pages 164 – 177. Springer-Verlag GmbH, August 2003.

[GMF04] R. Guha, R. Mccool, and R. Fikes. Contexts for the semantic web. In *International Semantic Web Conference, volume 3298 of Lecture Notes in Computer Science*, pages 32–46. Springer, 2004.

[Goe04] Brian Goetz. Java theory and practice: Dynamic compilation and performance measurement, 2004.

[Goe05] Brian Goetz. Java theory and practice: Anatomy of a flawed microbenchmark. Technical report, IBM developerWorks - Technical Report, 2005.

[GR92] Jim Gray and Andreas Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 1992.

[Gre01] Saul Greenberg. Context as a dynamic construct. *Hum.-Comput. Interact.*, 16(2):257–268, 2001.

[Gru95] Thomas R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *Int. J. Hum.-Comput. Stud.*, 43(5-6):907–928, 1995.

[GS91] F. Gomez and C. Segami. Classification-based reasoning. *Systems, Man and Cybernetics, IEEE Transactions on*, 21(3):644 –659, 1991.

[GSB02] Hans W. Gellersen, Albercht Schmidt, and Michael Beigl. Multi-sensor context-awareness in mobile devices and smart artifacts. *Mobile Networks and Applications*, 7(5):341–351, 2002.

[GSB08] Avelino J. Gonzalez, Brian S. Stensrud, and Gilbert Barrett. Formalizing context-based reasoning: A modeling paradigm for representing tactical human behavior. *Int. J. Intell. Syst.*, 23:822–847, July 2008.

[GSS02] David Garlan, Daniel P. Siewiorek, and Peter Steenkiste. Project aura: Toward distraction-free pervasive computing. *IEEE Pervasive Computing*, 1:22–31, 2002.

[Har09] Olaf Hartig. Querying trust in rdf data with tsparql. In *6th Annual European Semantic Web Conference (ESWC2009)*, pages 5–20, June 2009.

[HB11] Tom Heath and Christian Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Morgan & Claypool, 1st edition, 2011.

[HBS08] Alice Hertel, Jeen Broekstra, and Heiner Stuckenschmidt. RDF storage and retrieval systems. On-line, 2008.

[HDM05] Tom Heath, Martin Dzbor, and Enrico Motta. Supporting user tasks and context: Challenges for semantic web research. *Proc. ESWC2005 Workshop on End-User Aspects of the Semantic Web (UserSWeb)*, 2005.

[HDW09] Dexter H. Hu, Fan Dong, and Cho-Li Wang. A semantic context management framework on mobile device. In *ICESS '09: Proceedings of the 2009 International Conference on Embedded Software and Systems*, pages 331–338, Washington, DC, USA, 2009. IEEE Computer Society.

[HG03] Stephen Harris and Nicholas Gibbins. 3store: Efficient bulk rdf storage. In *Practical and Scalable Semantic Systems, Proceedings of the First International Workshop on Practical and Scalable Semantic Systems*, 2003.

[HGK+04] Lars Erik Holmquist, Hans-Werner Gellersen, Gerd Kortuem, Albrecht Schmidt, Martin Strohbach, Stavros Antifakos, Florian Michahelles, Bernt Schiele, Michael Beigl, and Ramia Mazé. Building intelligent environments with smart-its. *IEEE Comput. Graph. Appl.*, 24:56–64, January 2004.

[Him] Michael Himsolt. Gml: A portable graph file format. Technical report, Universität Passau, 94030 Passau, Germany.

[HIMB05] K. Henricksen, J. Indulska, T. McFadden, and S. Balasubramaniam. Middleware for Distributed Context-Aware Systems. In Robert Meersman and Zahir Tari, editors, *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA. Proceedings of the OTM Confederated International Conferences: CoopIS, DOA and ODBASE 2005, Part 1. International Symposium on Distributed Objects and Applications (DOA), Agia Napa, Cyprus*, pages 846–863. Springer, 2005.

[HIR02] Karen Henricksen, Jadwiga Indulska, and Andry Rakotonirainy. Modeling context information in pervasive computing systems. In *Pervasive '02: Proceedings of the First International Conference on Pervasive Computing*, pages 167–180, London, UK, 2002. Springer-Verlag.

[HKM+10] Sayed Y. Hashimi, Satya Komatineni, Dave MacLean, Sayed Y. Hashimi, Satya Komatineni, and Dave MacLean. *Pro Android 2*. Apress, 2010.

[HKS06] Mike Hazas, John Krumm, and Thomas Strang, editors. *Location- and Context-Awareness, Second International Workshop, LoCA 2006, Dublin, Ireland, May 10-11, 2006, Proceedings*, volume 3987 of *Lecture Notes in Computer Science*. Springer, 2006.

[HM01] Volker Haarslev and Ralf Möller. RACER System Description. In R. Goré, A. Leitsch, and T. Nipkow, editors, *Automated Reasoning: First International Joint Conference (IJCAR) 2001*, volume 2083 of *Lecture Notes in Computer Science*, page 701, Siena, Italy, June18-23 2001. Springer-Verlag.

[HM04] Patrick Hayes and Brian McBride. Rdf semantics, 2 2004.

[HM05] C. Huebscher and A. McCann. An adaptive middleware framework for context-aware applications. *Personal Ubiquitous Comput.*, 10(1):12–20, 2005.

[HM08] Tom Heath and Enrico Motta. Revyu: Linking reviews and ratings into the web of data. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, 6:266–273, November 2008.

[HMD05] Tom Heath, Enrico Motta, and Martin Dzbor. Context as a foundation for a semantic desktop. In Stefan Decker, Jack Park, Dennis Quan, and Leo Sauermann, editors, *Proc. of Semantic Desktop Workshop at the ISWC, Galway, Ireland, November 6*, volume 175, November 2005.

[HMPR04] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *Management Information Systems Quarterly*, 28(1):75–106, 2004.

[HNBr97] Richard Hull, Philip Neaves, and James Bedford-roberts. Towards situated computing. In *In Proceedings of The First International Symposium on Wearable Computers*, pages 146–153, 1997.

[HPSB+04] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. Swrl: A semantic web rule language combining owl and ruleml, May 2004.

[HRSZ11] Bernhard Haslhofer, Elaheh Momeni Roochi, Bernhard Schandl, and Stefan Zander. Europeana rdf store report. Technical report, University of Vienna, Vienna, March 2011.

[HS03] Hagen Höpfner and Kai-Uwe Sattler. Semantic Replication in Mobile Federated Information Systems. In Anne E. James, Stefan Conrad, and Wilhelm Hasselbring, editors, *Engineering Federated Information Systems, Proceedings of the 5th Workshop EFIS 2003, Coventry, UK*, pages 36–41. aka / IOS Press / infix, 2003.

[HSM+10] Wolfgang Halb, Alexander Stocker, Harald Mayer, Helmut Mülner, and Ilir Ademi. Towards a commercial adoption of linked open data for online content providers. In *Proceedings of the 6th International Conference on Semantic Systems*, I-SEMANTICS '10, pages 16:1–16:8, New York, NY, USA, 2010. ACM.

[HSMY08] Li Han, Jyri P. Salomaa, Jian Ma, and Kuifei Yu. Research on context-aware mobile computing. In *AINA Workshops*, pages 24–30. IEEE Computer Society, 2008.

[HSP+03] Thomas Hofer, Wieland Schwinger, Mario Pichler, Gerhard Leonhartsberger, Josef Altmann, and Werner Retschitzegger. Context-awareness on mobile devices - the hydrogen approach. In *HICSS '03: Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03) - Track 9*, page 292.1, Washington, DC, USA, 2003. IEEE Computer Society.

[HSW94] Yixiu Huang, Prasad Sistla, and Ouri Wolfson. Data Replication for Mobile Computers. In Richard Thomas Snodgrass and Marianne Winslett, editors, *Proceedings of the 1994 ACM SIGMOD international conference on Management of data (SIGMOD '94)*, pages 13–24, New York, NY, USA, 1994. ACM.

[IRRH03] Jadwiga Indulska, Ricky Robinson, Andry Rakotonirainy, and Karen Henricksen. Experiences in using cc/pp in context-aware systems. In *In Proc. of the Intl. Conf. on Mobile Data Management (MDM*, pages 247–261. Springer, 2003.

[IS03] Jadwiga Indulska and Peter Sutton. Location management in pervasive systems. In *ACSW Frontiers '03: Proceedings of the Australasian information security workshop conference on ACSW frontiers 2003*, pages 143–151, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc.

[JW04] Ian Jacobs and Norman Walsh. *Architecture of the World Wide Web, Volume One - W3C Recommendation 15 December 2004.* World Wide Web Consortium, December 2004.

[KA04] Manasawee Kaenampornpan and Bath Ba Ay. An intergrated context model: Bringing activity to context. In *In Workshop on Advanced Context Modelling, Reasoning and Management - UbiComp*, 2004.

[KAC+02] G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, and M. Schol. RQL: A Declarative Query Language for RDF. In *Proceedings of the Eleventh International World Wide Web Conference (WWW'02)*, Honolulu, Hawaii, USA, May7-11 2002.

[KB06] Maryam Kamvar and Shumeet Baluja. A large scale study of wireless search behavior: Google mobile search. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 701–709, New York, NY, USA, 2006. ACM.

[KC04] Graham Klyne and Jeremy J. Carroll. *Resource Description Framework (RDF): Concepts and Abstract Syntax (W3C Recommendation 10 February 2004).* World Wide Web Consortium, 2004.

[Kel06] Diane Kelly. Evaluating personal information management behaviors and tools. *Commun. ACM*, 49:84–86, January 2006.

[KHK+04] Panu Korpipää, Jonna Häkkilä, Juha Kela, Sami Ronkainen, and Ilkka Känsälä. Utilising context ontology in mobile device application personalisation. In *Proceedings of the 3rd international conference on Mobile and ubiquitous multimedia*, MUM '04, pages 133–140, New York, NY, USA, 2004. ACM.

[Kis07a] Cédric Kiss. Composite capabilities/preference profiles: Structure and vocabularies 2.0, 2007.

[Kis07b] Cédric Kiss. Composite capability/preference profiles (cc/pp): Structure and vocabularies 2.0. World Wide Web Consortium, Working Draft WD-CCPP-struct-vocab2-20070430, May 2007.

[Kja07] Kristian Ellebaek Kjaer. A survey of context-aware middleware. In *SE'07: Proceedings of the 25th conference on IASTED International Multi-Conference*, pages 148–155, Anaheim, CA, USA, 2007. ACTA Press.

[KLO+04] John Krogstie, Kalle Lyytinen, Andreas Lothe Opdahl, Barbara Pernici, Keng Siau, and Kari Smolander. Research areas and challenges for mobile information systems. *International Journal of Mobile Communications*, 2(3):220–234, 2004.

[KMK+03] P. Korpipaa, J. Mantyjarvi, J. Kela, H. Keranen, and E.J. Malm. Managing context information in mobile devices. *Pervasive Computing, IEEE*, 2(3):42–51, July-Sept. 2003.

[KMS+05] Panu Korpipää, Esko-Juhani Malm, Ilkka Salminen, Tapani Rantakokko, Vesa Kyllönen, and Ilkka Känsälä. Context management for end user development of context-aware applications. In *MDM '05: Proceedings of the 6th international conference on Mobile data management*, pages 304–308, New York, NY, USA, 2005. ACM.

[KPL+04] Sung Woo Kim, Sang Hyun Park, JungBong Lee, Young Kyu Jin, Hyun-Mi Park, Amy Chung, SeungEok Choi, and Woo Sik Choi. Sensible appliances: applying context-awareness to appliance design. *Personal Ubiquitous Comput.*, 8(3-4):184–191, 2004.

[KQJH03] B. Kummerfeld, A. Quigley, C. Johnson, and R. Hexel. Merino: Towards an intelligent environment architecture for multi-granularity context description. In K. Cheverst, N. de Carolis, and A. Kruger, editors, *Online Proceedings of the UM (User Modeling) 2003 Workshop on User Modeling for Ubiquitous Computing*, pages 29–35, 2003.

[KRW09] Carsten Ke, Martin Raubal, and Christoph Wosniok. Semantic rules for context-aware geographical information retrieval. In *Proceedings of the 4th European conference on Smart sensing and context*, EuroSSC'09, pages 77–92, Berlin, Heidelberg, 2009. Springer-Verlag.

[KTCY09] Yung-Wei Kao, Ching-Tsorng Tsai, Tung-Hing Chow, and Shyan-Ming Yuan. An offline browsing system for mobile devices. In *Proceedings of the 11th International Conference on Information Integration and Web-based Applications & Services*, iiWAS '09, pages 338–343, New York, NY, USA, 2009. ACM.

[Kuu95] Kari Kuutti. *Activity theory as a potential framework for human-computer interaction research*, pages 17–44. Massachusetts Institute of Technology, Cambridge, MA, USA, 1995.

[LBK+09] Jens Lehmann, Chris Bizer, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. DBpedia - a crystallization point for the web of data. *Journal of Web Semantics*, 7(3):154–165, 2009.

[LBWK05] Marko Luther, Sebastian Böhm, Matthias Wagner, and Johan Koolwaaij. Enhanced Presence Tracking for Mobile Applications. In *ISWC'05 Demo Track*, 2005.

[Len96] Richard Lenz. Adaptive distributed data management with weak consistent replicated data. In *SAC '96: Proceedings of the 1996 ACM symposium on Applied Computing*, pages 178–185, New York, NY, USA, 1996. ACM.

[Lew07] Rhys Lewis. Dereferencing http uris, May 2007.

[LFWK08] Marko Luther, Yusuke Fukazawa, Matthias Wagner, and Shoji Kurakake. Situational reasoning for task-oriented mobile service recommendation. *The Knowledge Engineering Review*, 23(1):7–19, 2008.

[LMWK05] M. Luther, B. Mrohs, S. Wagner, M.and Steglich, and W. Kellerer. Situational reasoning-a practical owl use case. In *Proceedings of the 7th International Symposium on Autonomous Decentralized Systems (ISADS'05)*, pages 96–103, Chengdu, China, 2005.

[LPPRH10] Danh Le-Phuoc, Josiane X. Parreira, Vinny Reynolds, and Manfred Hauswirth. RDF on the go: An RDF storage and query processor for mobile devices. In *9th International Semantic Web Conference (ISWC2010)*, November 2010.

[LPS86] Doug Lenat, Mayank Prakash, and Mary Shepherd. Cyc: Using common sense knowledge to overcome brittleness and knowledge acquistion bottlenecks. *AI Mag.*, 6:65–85, January 1986.

[LSD+02] Hui Lei, Daby M. Sow, John S. Davis, II, Guruduth Banavar, and Maria R. Ebling. The design and applications of a context service. *SIGMOBILE Mob. Comput. Commun. Rev.*, 6:45–55, October 2002.

[Mad08] Gerald Madlmayr. A mobile trusted computing architecture for a near field communication ecosystem. In *Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services*, iiWAS '08, pages 563–566, New York, NY, USA, 2008. ACM.

[MBDH02] Jayant Madhavan, Philip A. Bernstein, Pedro Domingos, and Alon Y. Halevy. Representing and reasoning about mappings between domain models. In *Eighteenth national conference on Artificial intelligence*, pages 80–86, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence.

[MBH+04] David Martin, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, Sheila McIlraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry R. Payne, Evren Sirin, Naveen Srinivasan, and Katia Sycara. Owl-s: Semantic markup for web services. 2004.

[MC02] René Meier and Vinny Cahill. Steam: Event-based middleware for wireless ad hoc network. In *ICDCSW '02: Proceedings of the 22nd International Conference on Distributed Computing Systems*, pages 639–644, Washington, DC, USA, 2002. IEEE Computer Society.

[McB01] Brian McBride. Jena: Implementing the RDF model and syntax specification. In *Proceedings of the 2nd International Workshop on the Semantic Web.*, Hongkong, May 1 2001.

[McL02] Jay F McLain. Offline viewing of internet content with a mobile device - united states patent 6493758. United States of America Patent and Trademark Office. Granted Patents (USPTO), Application Number: 9/149694, 12 2002.

[MD02] M. Mealling and R. Denenberg. Uniform resource identifiers (uris), urls, and uniform resource names (urns): Clarifications and recommendations. Technical report, Joint W3C/IETF URI Planning Interest Group - Network Working Group, August 2002.

[Mei10] Reto Meier. *Professional Android 2 application development.* Wiley Pub., 2010.

[MFC07] Jennifer Munnelly, Serena Fritsch, and Siobhan Clarke. An aspect-oriented approach to the modularisation of context. In *PERCOM '07: Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications*, pages 114–124, Washington, DC, USA, 2007. IEEE Computer Society.

[MM04] Frank Manola and Eric Miller, editors. *RDF Primer*. W3C Recommendation. World Wide Web Consortium, February 2004.

[MMG02] M. Lynne Markus, Ann Majchrzak, and Les Gasser. A design theory for systems that support emergent knowledge processes. *MIS Q.*, 26(3):179–212, September 2002.

[MPSP09] Boris Motik, Peter F. Patel-Schneider, and Bijan Parsia. *OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (W3C Recommendation 27 October 2009)*. World Wide Web Consortium, October 2009. Available at `http://www.w3.org/TR/owl2-syntax/`.

[MRS08] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, Juli 2008.

[MS95] Salvatore T. March and Gerald F. Smith. Design and natural science research on information technology. *Decis. Support Syst.*, 15(4):251–266, December 1995.

[MSLN00] Ethan Miller, Dan Shen, Junli Liu, and Charles Nicholas. Performance and scalability of a large-scale n-gram based information retrieval system. *Journal of Digital Information*, 1, 2000.

[MT07] Kristijan Mihalic and Manfred Tscheligi. 'divert: mother-in-law': representing and evaluating social context on mobile devices. In *MobileHCI '07: Proceedings of the 9th international conference on Human computer interaction with mobile devices and services*, pages 257–264, New York, NY, USA, 2007. ACM.

[MTD08] Atif Manzoor, Hong-Linh Truong, and Schahram Dustdar. On the evaluation of quality of context. In *EuroSSC '08: Proceedings of the 3rd European Conference on Smart Sensing and Context*, pages 140–153, Berlin, Heidelberg, 2008. Springer-Verlag.

[MWL+08] Li Ma, Chen Wang, Jing Lu, Feng Cao, Yue Pan, and Yong Yu. Effective and efficient semantic web data management over db2. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 1183–1194, New York, NY, USA, 2008. ACM.

[MYS05] Junfeng Man, Aimin Yang, and Xingming Sun. Shared ontology for pervasive computing. In Stephane Grumbach, Liying Sui, and Victor Vianu, editors, *Advances in Computer Science – ASIAN 2005. Data Management on the Web*, volume 3818 of *Lecture Notes in Computer Science*, pages 64–78. Springer Berlin / Heidelberg, 2005. 10.1007/115963707.

[N3P] Primer: Getting into rdf & semantic web using n3.

[Nar95] Bonnie A. Nardi. *Context and Consciousness: Activity Theory and Human-Computer Interaction*. The MIT Press, Cambridge, MA, first edition, November 1995.

[Noy04] Natalya F. Noy. Semantic integration: a survey of ontology-based approaches. *SIGMOD Rec.*, 33:65–70, December 2004.

[NP01] Ian Niles and Adam Pease. Towards a standard upper ontology. In Chris Welty and Barry Smith, editors, *In Proceedings of the 2nd International Conference on Formal Ontology in Information Systems(FOIS-2001), Chris Welty and Barry Smith*, 2001.

[OA98] Pinar Öztürk and Agnar Aamodt. A context model for knowledge-intensive case-based reasoning. *Int. J. Hum.-Comput. Stud.*, 48(3):331–355, 1998.

[ODC+08] Eyal Oren, Renaud Delbru, Michele Catasta, Richard Cyganiak, Holger Stenzhorn, and Giovanni Tummarello. Sindice.com: A document-oriented lookup index for open linked data. *International Journal of Metadata, Semantics and Ontologies*, 3(1), 2008.

[Owe09] Alisdair Owens. An investigation into improving rdf store performance, 2009.

[OWL04] Owl web ontology language overview. W3c recommendation, World Wide Web Consortium, February 2004.

[PB03] Paul Prekop and Mark Burnett. Activities, context and ubiquitous computing. *Computer Communications*, 26(11):1168 – 1176, 2003. Ubiquitous Computing.

[PBKL06] Emmanuel Pietriga, Christian Bizer, David Karger, and Ryan Lee. Fresnel: A Browser-Independent presentation vocabulary for RDF. In *The Semantic Web - ISWC 2006*, volume 4273 of *Lecture Notes in Computer Science*, pages 158–171. Springer-Verlag, 2006.

[PdBW+04] Davy Preuveneers, Jan Van den Bergh, Dennis Wagelaar, Andy Georges, Peter Rigole, Tim Clerckx, Yolande Berbers, Karin Coninx, Viviane Jonckers, and Koen De Bosschere. Towards an extensible context ontology for ambient intelligence. In Panos Markopoulos, Berry Eggen, Emile Aarts, and James L. Crowley, editors, *Second European Symposium on Ambient Intelligence*, volume 3295 of *LNCS*, pages 148 – 159, Eindhoven, The Netherlands, Nov 8 – 11 2004. Springer.

[Per06] Barbara Pernici, editor. *Mobile Information Systems: Infrastructure and Design for Adaptivity and Flexibility*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[PFFC09] Panagiotis Pediaditis, Giorgos Flouris, Irini Fundulaki, and Vassilis Christophides. On explicit provenance management in rdf/s graphs. In James Cheney, editor, *Workshop on the Theory and Practice of Provenance*. USENIX, 2009.

[PG01] Eric Prud'hommeaux and Benjamin Grosof. Rdf query survey, 2001.

[Pre04] Jan; et.Al. Preuveneers, Davy; Van den Bergh. Towards an extensible context ontology for ambient intelligence. *Lecture Notes in Computer Science*, Volume 3295/2004:148–159, 2004.

[PRS06] Carsten Pils, Ioanna Roussaki, and Maria Strimpakou. Location-based context retrieval and filtering. In *LoCA*, pages 256–273, 2006.

[PS08a] Eric Prud'hommeaux and Andy Seaborne. *SPARQL Query Language for RDF*. W3C Consortium, http://www.w3.org/TR/rdf-sparql-query/, w3c recommendation edition, January 2008.

[PS08b] Eric Prud'hommeaux and Andy Seaborne. *SPARQL Query Language for RDF (W3C Recommendation 15 January 2008)*. World Wide Web Consortium, 2008.

[PSHH04] Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks. Owl web ontology language semantics and abstract syntax. Technical report, W3C Recommendation, February 2004.

[PvHS07] P. Pawar, A. T. van Halteren, and K. Sheikh. Enabling context-aware computing for the nomadic mobile user: A service oriented and quality driven approach. In *Proceedings of IEEE Wireless Communications and Networking Conference, 2007.WCNC 2007., Hong Kong, China*, pages 2531–2536. IEEE Communication Society, March 2007.

[REB+06] F. Ramparany, J. Euzenat, T. H. F. Broens, A. Bottaro, and R. Poortinga. Context management and semantic modelling for ambient intelligence. Technical Report TR-CTIT-06-52, Enschede, April 2006.

[RF05] P. Remagnino and G.L. Foresti. Ambient intelligence: A new multidisciplinary paradigm. *Systems, Man and Cybernetics, Part A, IEEE Transactions on*, 35(1):1–6, Jan. 2005.

[RHP+10] Vinny Reynolds, Michael Hausenblas, Axel Polleres, Manfred Hauswirth, and Vinod Hegde. Exploiting linked open data for mobile augmented reality. In *W3C Workshop: Augmented Reality on the Web*, June 2010.

[RLMM09] Rick Rogers, John Lombardo, Zigurd Mednieks, and Blake Meike. *Android Application Development: Programming with the Google SDK*. O'Reilly, Beijing, 2009.

[RPM98] N. S. Ryan, J. Pascoe, and D. R. Morse. Enhanced reality fieldwork: the context-aware archaeological assistant. In V. Gaffney, M. van Leusen, and S. Exxon, editors, *Computer Applications in Archaeology 1997*, British Archaeological Reports, Oxford, October 1998. Tempus Reparatum.

[RS97] E. Roche and Y. Schabes, editors. *Finite-State Language Processing*. MIT Press, Cambridge, MA, 1997.

[RSP07] Ioanna Roussaki, Maria Strimpakou, and Carsten Pils. Distributed context retrieval and consistency control in pervasive computing. *J. Netw. Syst. Manage.*, 15(1):57–74, 2007.

[RTA05] Dimitrios Raptis, Nikolaos Tselios, and Nikolaos Avouris. Context-based design of mobile applications for museums: a survey of existing practices. In *MobileHCI '05: Proceedings of the 7th international conference on Human computer interaction with mobile devices & services*, pages 153–160, New York, NY, USA, 2005. ACM.

[RVW05] Philip Robinson, Harald Vogt, and Waleed Wagealla, editors. *Privacy, Security and Trust within the Context of Pervasive Computing*. The Kluwer International Series in Engineering and Computer Science. Springer Science+Business Media, Inc., 2005.

[RW03] Janice Redish and Dennis Wixon. The human-computer interaction handbook. chapter Task analysis, pages 922–940. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 2003.

[Sat02] M. Satyanarayanan. Pervasive computing: vision and challenges. *Personal Communications, IEEE*, 8(4):10–17, August 2002.

[SB08a] Robert Schmohl and Uwe Baumgarten. Context-aware computing: a survey preparing a generalized approach. In *Proceedings of the International MultiConference of Engineers and Computer Scientists 2008*, volume Vol. 1, Hong Kong, 2008.

[SB08b] Robert Schmohl and Uwe Baumgarten. A generalized context-aware architecture in heterogeneous mobile computing environments. In *ICWMC '08: Proceedings of the 2008 The Fourth International Conference on Wireless and Mobile Communications*, pages 118–124, Washington, DC, USA, 2008. IEEE Computer Society.

[SBD05] Leo Sauermann, Ansgar Bernardi, and Andreas Dengel. Overview and outlook on the semantic desktop. In Stefan Decker, Jack Park, Dennis Quan, and Leo Sauermann, editors, *Proceedings of the 1st Workshop on The Semantic Desktop at the ISWC 2005 Conference*, volume 175 of *CEUR-WS.org*, pages 1 – 18. CEUR-WS, November 2005.

[SBwG98] Albrecht Schmidt, Michael Beigl, and Hans w. Gellersen. There is more to context than location. *Computers and Graphics*, 23:893–901, 1998.

[SC08] Leo Sauermann and Richard Cyganiak. Cool uris for the semantic web. W3C Interest Group Note, December 2008.

[Sch10] Bernhard Schandl. Replication and Versioning of Partial RDF Graphs. In *Proceedings of the 7th European Semantic Web Conference (ESWC 2010)*, 2010.

[Sea04] Andy Seaborne. Rdql – a query language for rdf. W3C Member Submission, http://www.w3.org/Submission/2004/SUBMRDQL-20040109/, Jan. 2004.

[SFH09] Holger Schmidt, Florian Flerlage, and Franz J. Hauck. A generic context service for ubiquitous environments. In *Proceedings of the 2009 IEEE International Conference on Pervasive Computing and Communications*, pages 1–6, Washington, DC, USA, 2009. IEEE Computer Society.

[SKSB09] Leo Sauermann, Malte Kiesel, Kinga Schumacher, and Ansgar Bernardi. *Semantic Desktop*, pages 337–362. Springer, 2009.

[SKW07] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. YAGO: a core of semantic knowledge unifying WordNet and Wikipedia. In *WWW '07: Proceedings of the 16th International World Wide Web Conference, Banff, Canada*, pages 697–706, 2007.

[SLGH08] Timothy Sohn, Kevin A. Li, William G. Griswold, and James D. Hollan. A diary study of mobile information needs. In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 433–442, New York, NY, USA, 2008. ACM.

[SLPF03] Thomas Strang, Claudia Linnhoff-Popien, and Korbinian Frank. Cool: A context ontology language to enable contextual interoperability. In *LNCS 2893: Proceedings of 4th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS2003). Volume 2893 of Lecture Notes in Computer Science (LNCS)., Paris/France*, pages 236–247. Springer Verlag, 2003.

[SP04] Thomas Strang and Claudia L. Popien. A context modeling survey. In *UbiComp 1st International Workshop on Advanced Context Modelling, Reasoning and Management*, pages 31–41, Nottingham, September 2004.

[SPG+07] E. Sirin, B. Parsia, B. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical owl-dl reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):51–53, June 2007.

[SS00] Albrecht Schmidt and Albrecht Schmidt. Implicit human computer interaction through context. Technical report, Personal Technologies, 2000.

[sSO+05] m. c. schraefel, Daniel A. Smith, Alisdair Owens, Alistair Russell, Craig Harris, and Max Wilson. The evolving mspace platform: leveraging the semantic web on the trail of the memex. In *HYPERTEXT '05: Proceedings of the sixteenth ACM conference on Hypertext and hypermedia*, pages 174–183, New York, NY, USA, 2005. ACM.

[ST94] B.N. Schilit and M.M. Theimer. Disseminating active map information to mobile hosts. *Network, IEEE*, 8(5):22–32, Sep/Oct 1994.

[STS98] Kerstin Schwarz, Can Türker, and Gunter Saake. Analyzing and formalizing dependencies in generalized transaction structures. In *In Proceedings of the International Workshop on Issues and Applications of Database Technology (IADT'98*, pages 6–9, 1998.

[SVLO+11] M. Strobbe, O. Van Laere, F. Ongenae, S. Dauwe, B. Dhoedt, F. De Turck, P. Demeester, and K. Luyten. Integrating location and context information for novel personalised applications. *Pervasive Computing, IEEE*, PP(99):1, 2011.

[SWB+08] Thomas Springer, Patrick Wustmann, Iris Braun, Waltenegus Dargie, and Michael Berger. A comprehensive approach for situation-awareness based on sensing and reasoning about context. In *UIC '08: Proceedings of the 5th international conference on Ubiquitous Intelligence and Computing*, pages 143–157, Berlin, Heidelberg, 2008. Springer-Verlag.

[SWvS07] Kamran Sheikh, Maarten Wegdam, and Marten van Sinderen. Middleware support for quality of context in pervasive context-aware systems. In *PERCOMW '07: Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 461–466, Washington, DC, USA, 2007. IEEE Computer Society.

[SZ09a] Bernhard Schandl and Stefan Zander. Adaptive rdf graph replication for mobile semantic web applications. *Ubiquitous Computing and Communication Journal (Special Issue on Managing Data with Mobile Devices)*, -(-), July 2009.

[SZ09b] Bernhard Schandl and Stefan Zander. A framework for adaptive rdf graph replication for mobile semantic web applications. In *Joint Workshop on Advanced Technologies and Techniques for Enterprise Information Systems (Session on Managing Data with Mobile Devices)*, pages 154–163. INSTICC Press, May 2009.

[SZ10a] Bernhard Schandl and Stefan Zander. MobiSem System Documentation - Deliverable d1.7. Project deliverable, University of Vienna, September 2010.

[SZ10b] Bernhard Schandl and Stefan Zander. Synchronization and algorithm specification - deliverable d1.2. Project deliverable, University of Vienna, September 2010.

[Teo08] Hong-Siang Teo. An activity-driven model for context-awareness in mobile computing. In *MobileHCI '08: Proceedings of the 10th international conference on Human computer interaction with mobile devices and services*, pages 545–546, New York, NY, USA, 2008. ACM.

[TGE+06] Kieron R. Taylor, Robert J. Gledhill, Jonathan W. Essex, Jeremy G. Frey, Stephen W. Harris, and David De Roure. Bringing chemical data onto the semantic web. *Journal of Chemical Information and Modeling*, 46(3):939–952, 2006.

[TH06] D. Tsarkov and I. Horrocks. Fact++ description logic reasoner: System description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 292–297. Springer, 2006.

[THS09] Krishnaprasad Thirunarayan, Cory A. Henson, and Amit P. Sheth. Situation awareness via abductive reasoning from semantic sensor data: A preliminary report. *Collaborative Technologies and Systems, International Symposium on*, 0:111–118, 2009.

[TS09] Goce Trajcevski and Peter Scheuermann. Managing context evolution in pervasive environments. In *Proceedings of the 2nd International Conference on Pervasive Technologies Related to Assistive Environments*, PETRA '09, pages 19:1–19:2, New York, NY, USA, 2009. ACM.

[VBGK09] Julius Volz, Christian Bizer, Martin Gaedke, and Georgi Kobilarov. Discovering and maintaining links on the web of data. In *The Semantic Web – ISWC 2009: 8th International Semantic Web Conference, Chantilly, VA, USA*, pages 650–665. 2009.

[WB05] Ivo Widjaja and Sandrine Balbo. Spheres of role in context-awareness. In *OZCHI '05: Proceedings of the 17th Australia conference on Computer-Human Interaction*, pages 1–4, Narrabundah, Australia, Australia, 2005. Computer-Human Interaction Special Interest Group (CHISIG) of Australia.

[WBB08] Cathrin Weiss, Abraham Bernstein, and Sandro Boccuzzo. i-MoCo: Mobile Conference Guide - Storing and querying huge amounts of Semantic Web data on the iPhone/iPod Touch, October 2008.

[Wei91] Mark Weiser. The computer for the 21st century. *Scientific American*, (Communications, Computers, and Network), September 1991.

[Win01] Terry Winograd. Architectures for context. *Human-Computer Interaction*, 16(2):401–419, 2001.

[WJH97] Ouri Wolfson, Sushil Jajodia, and Yixiu Huang. An Adaptive Data Replication Algorithm. *ACM Trans. Database Syst.*, 22(2):255–314, 1997.

[WLL+07] Timo Weithöner, Thorsten Liebig, Marko Luther, Sebastian Böhm, Friedrich Henke, and Olaf Noppens. Real-world reasoning with owl. In *ESWC '07: Proceedings of the 4th European conference on The Semantic Web*, pages 296–310, Berlin, Heidelberg, 2007. Springer-Verlag.

[WRS+05] M. L. Wilson, A. Russell, D. A. Smith, A. Owens, and m. c. schraefel. mspace mobile: A mobile application for the semantic web. *End User Semantic Web Workshop, ISWC2005*, page 11, 2005.

[WVV+01] H. Wache, T. Vögele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hübner. Ontology-Based Integration of Information – A Survey of Existing Approaches. In *Proceedings of the IJCAI-01 Workshop on Ontologies and Information Sharing*, Seattle, USA, 2001.

[WW03] Shiow Yang Wu and Kun-Ta Wu. Dynamic Data Management for Location Based Services in Mobile Environments. In *7th International Database Engineering and Applications Symposium (IDEAS)*, pages 180–191. IEEE Computer Society, 2003.

[WX06] M. Wojciechowski and J. Xiong. Towards an open context infrastructure. *Proceedings of the Workshop on Context Awareness for Proactive Systems (CAPS'06)*, pages 125–136, 2006.

[WZGP04] Xiao Hang Wang, Da Qing Zhang, Tao Gu, and Hung Keng Pung. Ontology based context modeling and reasoning using owl. In *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, PERCOMW '04, pages 18–, Washington, DC, USA, 2004. IEEE Computer Society.

[XMS+05] Xing Xie, Gengxin Miao, Ruihua Song, Ji-Rong Wen, and Wei-Ying Ma. Efficient browsing of web search results on mobile devices based on block importance model. *Pervasive Computing and Communications, IEEE International Conference on*, 0:17–26, 2005.

[XYCS02] Fei Xia, Alex V. Yakovlev, Ian G. Clark, and Delong Shang. Data communication in systems with heterogeneous timing. *IEEE Micro*, 22:58–69, November 2002.

[YDM11] Juan Ye, Simon Dobson, and Susan McKeever. Situation identification techniques in pervasive computing: A review. *Pervasive and Mobile Computing*, In Press, Corrected Proof:–, 2011.

[YKIAL09] Jorma Ylinen, Mikko Koskela, Lari Iso-Anttila, and Pekka Loula. Near field communication network services. In *Third International Conference on the Digital Society (ICDS 2009)*, pages 89–93. IEEE Computer Society, IEEE Computer Society, 2009.

[Zan09] Stefan Zander. A context-aware approach for integrating semantic web technologies onto mobile devices. In *Proceedings of the 6th European Semantic Web Conference on The Semantic Web: Research and Applications*, ESWC 2009 Heraklion, pages 949–953, Berlin, Heidelberg, 2009. Springer-Verlag.

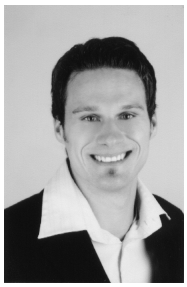
[ZS10] Stefan Zander and Bernhard Schandl. A Framework for Context-driven RDF Data Replication on Mobile Devices. In *Proceedings of the 6th International Conference on Semantic Systems (I-Semantics)*, Graz, Austria, 2010.

[ZS11] Stefan Zander and Bernhard Schandl. Context-driven rdf data replication on mobile devices. *Semantic Web Journal Special Issue on Real-time and Ubiquitous Social Semantics*, 1(1), 2011.

[ZS12a] Stefan Zander and Bernhard Schandl. Mobile devices, context, and linked data. to be published, 2012.

[ZS12b] Stefan Zander and Bernhard Schandl. *Semantic Web-enhanced Context-aware Computing in Mobile Systems: Principles and Application.* IGI Global, 1st edition, January 2012.

Curriculum Vitae



Stefan Zander

Diplom-Informatiker (FH), MSc, M.I.T.

University of Vienna
Liebiggasse 4/3-4
1010 Wien
Austria

T +43-1-4277-39646
F +43-1-4277-39640
stefan.zander@univie.ac.at
http://www.cs.univie.ac.at/stefan.zander

Private
Währinger Straße 57/12
1090 Wien
Austria

T +43-1-9138673
M +43-681-10457876
stefan_zander@gmx.de

Personal Data

Born at 5th Februar, 5th 1978 in Würzburg, Germany
Single, no children

Education

10/2008 – 06/2012
PhD in Computer Science, University of Vienna, Austria

03/2004 – 05/2007
Master degree in Business Administration (IT), University of Central Lancashire, England

10/2003 – 06/2006
Master degree in Organizational Development with IT, University of Applied Sciences Würzburg, Germany

10/1998 – 01/2004
Diploma in Computer Science, University of Applied Sciences Würzburg, Germany

Professional Experience

09/2010 – present
Research assistant, University of Vienna, Multimedia Information Systems Group

02/2007 – 08/2010
Research associate, University of Vienna, Department of Distributed and Multimedia Information Systems

10/2005 – 03/2006
Research associate in the EC-funded FP6 research project METOKIS

03/2002 – 11/2002
Collaboration with the Fraunhofer-Gesellschaft in a research project for the “High-Tech Offensive Zukunft Bayern”

09/2001 – 06/2002
Software developer, PASS IT-Consulting Group and Services, Höchberg, Frankfurt