Institute of Architecture of Application Systems
University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3139

Enabling Integration and Aggregation of Context Information into WS-BPEL Processes

Rodion Hagin

Course of Study:	Software Engineering
Examiner:	Prof. Dr. Frank Leymann
Supervisor:	Dipl.-Inf. Tobias Binz Dipl.-Inf. Steve Strauch
Commenced:	February 1, 2011
Completed:	August 2, 2011
CR-Classification:	H.3.3, H.3.4, H.4.1

Abstract

Previously, techniques of Context-Aware Computing were limited only to small scale monolithic applications due to the lack of standardized technologies which could support interoperability of services owned by different organizations. The advancement in *Service-Oriented Computing* technology allowed autonomous and heterogeneous applications to be exposed as Web Services and interconnected into service compositions exploiting well-agreed interfaces, protocols and message formats. The Web Service Business Process Execution Language (WS-BPEL) is the de-facto standard for composing reusable Web services. To enable handling of context information in applications, context information has to be made available within service compositions; hence, integrated in WS-BPEL processes. Through this means, new innovative context-enriched services can be built and provided using the convergence of context-aware computing and workflow technology. In this diploma thesis, context information provided by the *C-CAST Context Management Framework* and *Google Maps Web services*, is integrated into WS-BPEL, and business modelers are supported with the creation of context-based compositions. After surveying some of the current best practice and relevant literature in this area, this thesis presents a solution to this problem based on the *Integration Process Pattern* work previously done at the Institute of Architecture of Application Systems at the University of Stuttgart.

Contents

List of Figures

List of Tables

List of Listings

List of Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
ARC	Australian Research Council
ASR	Area Service Register
AWML	Augmented World Modeling Language
AWQL	Augmented World Query Language
BPM	Business Process Management
BPMN	Business Process Modeling Notation
C-CAST	Context Casting project
ccTLD	Top Level Domain (two-character value)
CEP	Complex Event Processing
CIPs	Context Integration Processes
CMF	Context Management Framework
ContextML	Context Modeling Language
EER	Enhanced Entity Relationship modeling
EMF	Eclipse Modeling Framework
ER	Entity Relationship modeling
ESB	Enterprise Service Bus
FTP	File Transfer Protocol
GEF	Graphical Editing Framework
HTTP	Hypertext Transfer Protocol
JAX-RS	Java API for RESTful Web services
JAX-WS	Java API for XML Web services
JBI	Java Business Integration
JMS	Java Message Service
JSON	JavaScript Object Notation
MDA	Model-driven Architecture
ODE	Orchestration Director Engine

ORM	Object-Role Modeling
OW2	ObjectWeb and Orientware Consortium
PaaS	Platform as a Service
R&D	Research and Development
REST	Representational State Transfer
SBPL	Standard Business Process Language
SWT	Standard Widget Tookit
UI	User Interface
UML	Unified Modelling Language
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WS	Web service
WS-BPEL	Web Service Business Process Execution Language
WSDL	Web Service Definition Language
WSN	Web Service Notification
WSRF	Web Service Resource Framework
XMI	XML Metadata Interchange
XML	eXtensible Markup Language
XSD	XML Schema Notation
XSLT	eXtensible Stylesheet Language Transformations

Chapter 1

Introduction

Cloud Computing [1], as an emerging network computing paradigm of distributed application environments is turning computation and service provisioning to a public utility. Combined with complementary Workflow technology, Cloud Computing enables service providers to enjoy elasticity, scalability, and high availability of computational resources while composing and orchestrating distributed business processes. It is of essential importance for service providers to focus on enhancing business processes by considering context information and how context awareness can maximize end users' experience and allow intelligent and advantageous adaptivity of service compositions.

The Web Service Business Process Execution Language (WS-BPEL) is the de-facto standard for composing reusable Web services [2]. To handle context information in applications the context information has to be made available within service compositions; hence, integrated into WS-BPEL processes. Through this means, new innovative context-enriched services can be built, and provided using the convergence of context-aware computing, workflow technology and the core tenets of cloud computing, e.g. by using the *4CaaSt* platform [3].

The EU project *4CaaSt* [3] aims to create an advanced PaaS Cloud platform which supports the optimized and elastic hosting of composite Internet-scale multi-tier applications. *4CaaSt* embeds all the necessary features easing programming of rich applications and enabling the creation of a true business ecosystem where applications coming from different providers can be tailored to different users, mashed up, and traded together.

This diploma thesis describes the concept of integrating context information into WS-BPEL service compositions in order to support workflow modelers to create context-aware compositions. Additionally, current work addresses the topic of aggregating of context information obtained from multiple data sources. After surveying some of the current best practice and relevant literature in this area, this thesis presents a solution to the outlined problems based on the *Integration Process Pattern* work previously done at the Institute of Architecture of Application Systems at the University of Stuttgart.

1.1. Motivating Example

In the beginning of this work a sample use case scenario is proposed and described in detail. The objective of this presentation is twofold: first, to give the reader a better understanding of how context information can be used in service compositions. Second, to develop a sample use case scenario used to evaluate the context integration and aggregation concepts realized at a later stage in this thesis.

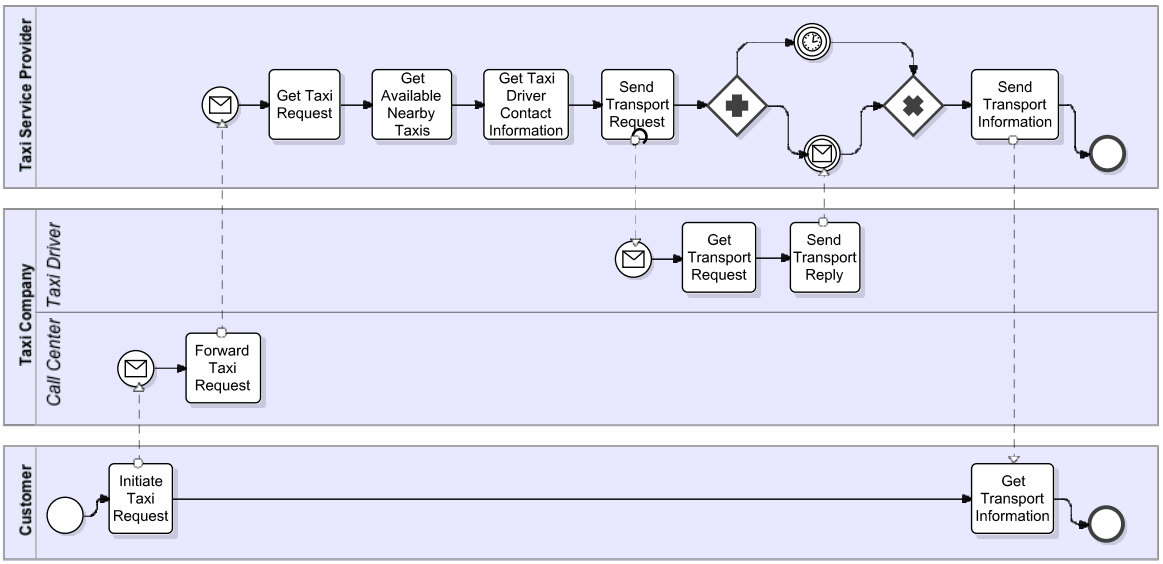

Figure 1.1.: Use Case Scenario – Taxi as a Service

The scenario used in this thesis is a taxi booking system using context information to offer taxi transport services tailored to best suit customer expectations. As illustrated in Figure 1.1, the booking request originates at the *Customer* side and contains information about origin address, destination address and other valuable information identifying this particular customer and her preferences. *Taxi Company* registers booking request which subsequently are forwarded to *Taxi Service Provider*. A *Taxi Service Provider* realizes a transport ordering process which can be shared among various taxi companies. In particular, the *Taxi Service Provider* process used in this sample use case scenario is responsible for the discovery of the nearest taxi cab to a customer. In order to achieve this, the service provider process extracts address information out of the booking request message, obtains the list of the available taxicabs next to the customer origin address, contacts a particular taxi driver to confirm booking and generates acknowledgment response for the *Taxi Company* after the taxi booking is accomplished.

Since the notion of context is very wide and includes many important aspects, it is preferable to set its thorough definition aside for the time being. With reference to the current scenario setting, context information can be represented as a sample set of context elements such as information identifying customer, taxi and their current locations. This kind of information can be leveraged in the sample use case scenario to achieve the following objectives:

- *Discovery of nearby taxi cabs* While a particular taxi company may incorporate multiple cities or locations across different regions and countries, the customer's current location details may be used to identify and forward transport booking requests to taxi cabs in the vicinity. As a result, taxi companies can provide customer transport service in the shortest time and minimize fuel expenses for their fleet of taxis. Additionally, transmitting booking requests to a restricted set of taxis rather than to all available taxis in the region will help reduce communication overhead.
- *Discovery of taxi contact details* Since taxi drivers working at a particular taxi company may share available taxi cabs, booking requests need to be forwarded to the taxi driver currently operating a particular taxi cab.

Furthermore, booking information specifying required destination address may help identify the preferable taxi cab type. For instance, a taxi cab with high baggage capacity hold might be more suitable for airport destinations. Additional information identifying customer and customer preferences can help provide special taxi transport services built to cater for instance to urgent corporate businesses, international tourist transport or transport of disabled persons.

Due to the fact that context knowledge represents information which is external to the business process and therefore is managed by external systems, a strategy is required to make context knowledge available in service compositions. It is worthy noting that context information can be obtained from multiple heterogeneous data sources which retrieve, model and disseminate context information in different ways. Hence, it is important to support aggregation and processing of context information from numerous heterogeneous data sources. The development of such integration and aggregation strategy is a subject of this diploma thesis. In the next section, main requirements to the context integration and aggregation approach are described in more detail.

1.2. Problem Statement

The main objective of this thesis is to evaluate existing concepts of context integration into service compositions to find the most suitable approach for future realization. A successful context integration and aggregation strategy should address the following list of problem domains.

Enabling integration of context in WS-BPEL compositions
In the last years Web Service Business Execution Language (WS-BPEL) has become the de-facto standard for composing and orchestrating Web services and is widely used in industry and research. Therefore, a general strategy is required to enable context integration into service compositions realized in WS-BPEL.

Providing a composition engine independent solution
The proposed context integration concept should rely on the non-extended WS-BPEL standard in order to ensure interoperability and composition engine independence.

Enabling encapsulation and outsourcing of context information aggregation
Business process modelers should be able to concentrate on modeling business logic and leverage context information in a simple and convenient way. High level context pre-processing and aggregation logic should be moved out and handled outside of the business process.

Providing integration support of different context provisioning systems
A major aspect is considered to be future application of the proposed approach to the integration of various context provisioning systems. Addressing this objective in the proposed integration approach enables high level aggregation of heterogeneous context data obtained from different data sources.

Enabling loosely coupling to context integration system
While interfaces of the integrated context provisioning systems may evolve over time, it is necessary to ensure that the proposed integration concept defines a necessary abstraction to the context provisioning system interfaces. The changes introduced to the interfaces should not imply the adaptation of business service compositions. This also allows the exchange of the underlying context provisioning systems without further modifications.

Providing component reusability
The software components of the proposed context integration and aggregation concept should be easily reusable for multiple context-aware service compositions.

Enabling integration system extensibility
The last, but equally important, aspect considers the extensibility of the proposed integration concept. It should be possible to integrate new context provisioning systems, define additional context aggregation components and model new context-aware service compositions.

In the rest of this diploma thesis different context integration strategies are introduced and compared against their complexity. Finally, a context strategy is selected which best suits the requirements outlined in this section.

1.3. Outline

The remainder of this thesis is structured into seven chapters. In detail, each chapter covers the following topics:

Chapter 2 - Fundamentals: Chapter 2 evaluates the research achievements over the past years in the area of context awareness and provides the current state of the art of context and integration of context into service compositions. The main attention of this chapter is focused on the presentation of the relevant context provisioning systems and comparison of the available approaches of context integration into service compositions.

Chapter 3 - Integration of Context Information: Chapter 3 describes the concept of the context information integration into service compositions. A context provisioning system *C-CAST CMF* introduced in the remainder of this thesis will be integrated into WS-BPEL compositions using the Integration Process Architecture Pattern approach.

Chapter 4 - Aggregation of Context Information: Chapter 4 is devoted to the aggregation of context information obtained from various context provisioning systems. This Chapter gives a general idea of context aggregation in service compositions, identifies aggregation components and describes an extension to the Integration Architecture Process Pattern approach.

Chapter 5 - Evaluation: Chapter 5 evaluates the concepts of context integration and aggregation realized in the previous chapters. The sample use case scenario is deployed to *Apache ODE* and *OW2 Orchestra* to ensure interoperability and engine implementation independency of the proposed integration solution.

Chapter 6 - Context Modeling Tool: Chapter 6 acts as the foundation for future implementation of an extension to WS-BPEL modeling tools to support modeling of context-aware service compositions. This chapter covers specification and design details and provides hands-on knowledge required for its implementation.

Chapter 7 - Summary and Future Work: The last chapter concludes this thesis and contains information about future work.

Chapter 2

Fundamentals and State of the Art

This chapter evaluates the research achievements over the past years in the area of context awareness. It gives the hands-on knowledge for better understanding of main concepts and principles by presenting historical development and the current state of the art of context and its integration into service compositions. The remaining sections of this chapter are structured as follows:

The first section of this chapter provides historical background information, discusses the fundamental contributions in this research area and explores the unique characteristics of context information. Furthermore, it examines in detail the basic components of the conceptual architecture model for context management.

The main attention of the second section is focused on the presentation of some of the relevant context provisioning systems and their services. The third section introduces different strategies to enable integration of context into service compositions. Finally, the available approaches are analyzed and compared against their complexity to find the most suitable approach for future realization.

2.1. Context

2.1.1. Definitions

General notion of Context

The definition of *Context* in computer science is primarily based on its common notion in the ordinary language. Derived from the Latin term *contextus* which means connecting or composing as by interweaving of parts, the word itself describes *Context* as the parts of information which immediately precede or follow any particular passage or text and determine its full meaning [4]. Within the field of computer science, this basic definition of *Context* was initially used by Noam Chomsky in his fundamental work describing the containment hierarchy of classes of formal grammars.

Since computing reflects the changes that society is undergoing, the definition of *Context* tends to get more general over time. According to the modern conventional interpretation the *Context* can be depicted as an active process dealing with the way humans weave their experience within their environment, to give it meaning [5]. This general interpretation of *Context* provides a reference point to present a further survey on context definition.

History of Context-Aware Computing

One of the first applications of context information in computing goes back to 1992 when Roy Want et al. introduced the *Active Badge Location System* [6]. A wearable tag designed in the form of an Active Badge transmitted information about the current location of an individual to a centralized location service, through a network of infrared sensors. The server in turn used the location information of humans to compute their proximity to the nearest telephone extension and forward telephone calls from the main switchboard. The idea of the Active Badge Location System was used as an image of the future of computing in the visionary article written for *Scientific American* by Mark Weiser [7]. Mark Weiser, thought of as the founder of the term *Ubiquitous Computing*, articulated the idea of invisible computation based on the principle that computer devices connected with strong communication networks vanish into the background and become transparently and seamlessly woven into the main activities of everyday life. According to him, location awareness is a technical issue of crucial importance. Provided with the information of their location and the location of objects in their environments, computer devices acquire a certain level of intelligence taking right proactive actions to adapt their computational behavior as per user expectations.

Mark Weiser's vision of the future of Information Technology had an enormous impact on the research. Adaptivity to the location of physical objects in the environment remained a key issue to achieve social sensitivity in computing, until Bill N. Schilit et al. wrapped location awareness into their notion of *Context-Aware Computing* [8] [9]. The new term was used to depict general software capability to adapt itself to the changing environment using context information. The aforementioned adaptation takes place according to the current location, identities of nearby individuals, hosts and accessible devices, as well as changes to those things over time. Though their context definition encompasses other attributes such as lighting, noise level, network connectivity, and communication costs, their primarily interpretation of context was based on location and proximity. Since the reach and relevance of the concept of context is much broader than what the proposed definition could capture, earlier researchers have tried to contribute to the context definition by suggesting various context elements and their classification.

Definition of Context

The notion of context has been extensively discussed in the literature. Manasawee Kaenampornpan et al. presented a survey on context definition issues, together with the discussion of different approaches [10]. They classified various context definitions regarding context elements such as location, infrastructure information, user information and activity, social aspects, time, device characteristics, and others. The results of their survey are shown in Table 2.1 where the rows depict the addressed context definitions and columns the elements of context, that researchers used to classify as part of their context.

The review of past research in context definition showed that researches had different views on what context is and which elements should be considered as part of context. To get detailed information on each suggested context definition, please refer to the survey [10]. Among the proposed definitions the one suggested by Gregory D. Abowd and Anind K. Dey is the most accurate [11]:

> *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.*

	Location	Conditions	Infrastructure (Computing environment)	Information on User	Social	User Activity	Time	Device Characteristics
Benerecetti et al. '01	**Physical Environment**			**Cultural Context**				
Schmidt et al '99	**Physical Environment**			**Human Factor**			✓	
Lieberman and Selker '00	**User Environment**	**Physical Environment**	✓	**User environment**			✓	
Hull et al. '97		**Physical Environment**		✓				✓
Chaimers and Sloman '99	✓		✓		✓	✓		✓
Lucas '01	**Physical Environment**		**Information Context**					✓
Schilit et al '94	**Physical Environment**		✓	**User Environment**				
Dey and Abowd '99	✓			**Identity**		✓	✓	**Identity**
Chen and Kotz '00	**Active / Passive**							

Table 2.1.: Context Classification Systems [10]

This definition is one of the most often cited definitions of context. The reason why this context interpretation supposed to be the most successful is well explained in the paper by Ahmed Soylu et al. [12]. In [13] A.K. Dey shows, that it is not possible to enumerate a complete set of context elements that cover context as a whole. A particular knowledge can be interpreted as a contextual information in one setting while it is not part of context in another setting. The notion of a new abstraction, defined as *situation* [13], eases the enumeration of context elements relevant for a specific application. Hence, rather than providing an exhaustive context definition, it is better to concentrate on the situational environment of a particular application and define all important aspects of a situation.

These necessary aspects can be determined by answering questions regarding the dimensions: *who*, *what*, *where*, *when* and *why* [13] [14] [15]. Other dimensions for classification of context aspects include *external* vs *internal* [16], as well as *physical* vs *logical* [17] dimensions according to the survey in [18]. The external dimension of context encompasses the elements of physical environments such as location, proximity, temperature and time, while the internal dimension includes information like user goals, tasks, emotional and physical state. Physical and logical dimensions differ in the way the context information is retrieved: physical context can be captured by hardware sensors, while logical context aggregates different kinds of context to enrich the semantics of this information. The extracted set of obtained context elements can be further reduced by considering the kind of a particular context class as defined by Guanling Chen and David Kotz in [19]. Context is classified in two kinds: *active* and *passive*. Active context is of crucial importance for a particular application since it influences its behavior. Passive context is in turn relevant, but not critical to an application.

Definition of Context-Awareness and Context-Aware Computing

As previously mentioned, the term *Context-Awareness* and *Context-Aware Computing* was initially introduced by Bill N. Schilit et al. in [8]:

> *Context-Aware computing is the ability of a mobile user's application to discover and react to changes in the environment they are situated in.*

Bill N. Schilit et al. categorize the capabilities of Context-Aware systems in:

- *Proximate Selection*, a user interface technique to ease the selection of the nearby located objects
- *Automatic Contextual Reconfiguration* of a system's structure in order to adapt to the changing environment
- *Commands* using contextual information to produce different results according to the context in which they are issued
- *Actions* triggered by changes in context and specified as simple IF-THEN rules

In a like manner, Richard Hull et al. defined *Situated Computing* as the ability of computing devices to detect, interpret, and respond to aspects of the user's local environment [20]. The main responsibilities of a system based on Situated Computing include data fusion and interpretation, event delivery about situational changes, as well as granting interfaces to query context information. Jason Pascoe described Context-Awareness by presenting a set of core capabilities of each context-aware system [21]. According to him, these capabilities include: *contextual sensing* of environmental states, *contextual adaptation* of computational behavior, *contextual discovery* of resources for their further utilization, and *contextual augmentation* of the environment with additional useful information.

The definition of *Context-Awareness* chosen by Anind K. Dey and Gregory D. Abowd differs from the others and is basically considered from the user's view [13].

> *A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task.*

Anind K. Dey and Gregory D. Abowd combined and simplified the ideas introduced by other researchers and proposed a different classification of context-aware features that context-aware applications may support: *presentation* of information and services to the user, automatic *execution* of a service, and *tagging* of context information for later retrieval. This most cited definition of Context-Awareness and context-aware features is supposed to be the fundamental contribution in the research area.

Context in Service Compositions

Previously, techniques of Context-Aware Computing were limited only to autonomous standalone applications due to the lack of standardized technologies which could support interoperability of services owned by different organizations. The advancement in *Service-Oriented Computing* (SOA) technology allowed autonomous and heterogeneous applications to be exposed as Web services and interconnected into service compositions leveraging well-agreed interfaces, protocols and message formats. Service-Oriented Computing technology enabled service compositions to leverage various types of context information and adapt their behavior according to the changes in the large scale, multi organizational environment. This turn in technology had an impact on the interpretation of the notion of Context and Context-Aware Computing.

Though Dey's definition of Context and Context-Aware Computing remained the most commonly cited among the research community, new definitions still emerged with a particular focus on interpretation of context specific to the domain of service compositions. Basic usage of context information can be classified as follows:

- *Data interpretation*
- *Service personalization*
- *Service process adaptation*
- *Automatic service execution*
- *Provisioning of relevant user-tailored services*

In [22], the authors use context to describe the underlying semantics of the exchanged data between Web services. Context is defined *as a collection of implicit assumptions that are required to obtain accurate data interpretation*.

In [23], context is defined to have its primarily use in *personalization of composite Web services*. The context elements are considered from a Web service's perspective and include information about user's environment, preferences, needs, and current execution status of a Web service. C. Chedira et el. [23] classify context in three various types: 1. *User Context* which encompasses a limited set of information about user's environment, preferences and needs; 2. *Web Service Context* which includes information about the current execution status of a Web service such as availability, and resource requirements; and 3. *Composite Context* which aggregates the context information of each Web service and exposes it for the whole service composition.

A. A. George describes context as *an environmental state which is external to a process, whose value can change independent of a process' lifecycle, and can influence the process' execution* [24]. The central idea articulates that a service process cannot directly change its context.

A common notion of Context is considered to be *the information characterizing the situation in which a Web service is being executed* [25]. Hence, a context-aware Web service is supposed to be cognizant of its situation and be able to use this information to provide relevant services or be executed or adapted automatically [26].

2.1.2. Context Management

A number of different approaches has been proposed to define a conceptual architecture model for context management. To present basic components of the architecture, this chapter initially outlines important actors which participate in the context management. Subsequently, various groups of basic components for context management are covered by adopting [12] and [27] as shown in Figure 2.1.

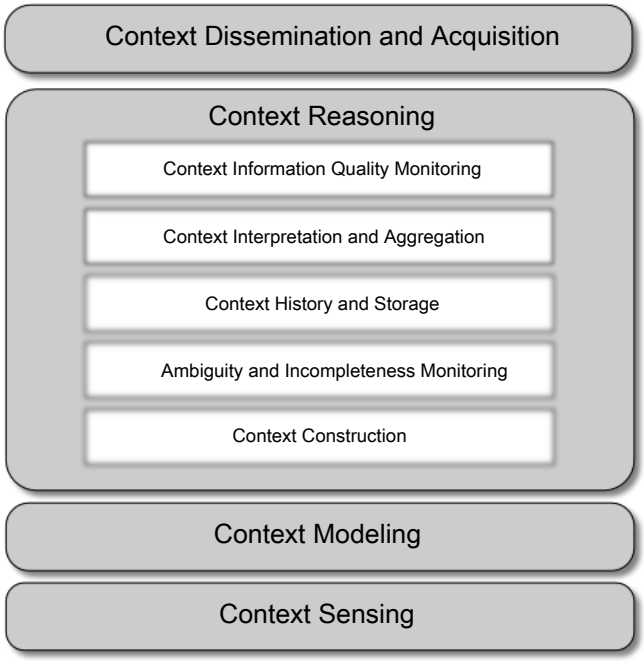

Figure 2.1.: Context Management Architecture – Main Components

Actors

Each context-aware system consists of software components with well defined roles. These roles are depicted in Figure 2.2 and described in detail as follows:

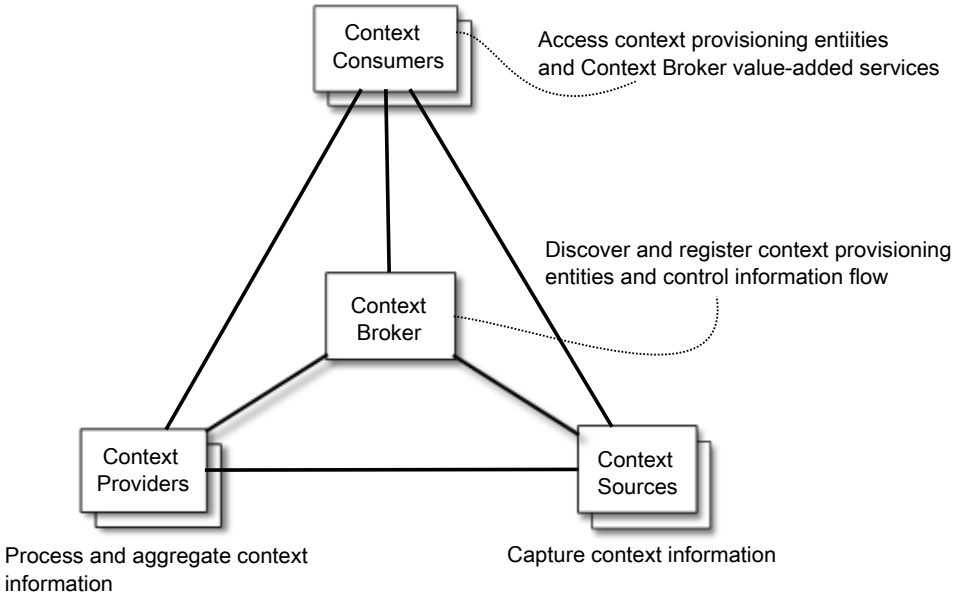

Figure 2.2.: Context Management Architecture – Actors

Context Source Local and remote data sources are usually wrapped into so-called Context Sources. In order to achieve interoperability between different system components a Context Source transforms fetched data in a well-defined format and exposes an interface for other components to access this data.

Context Provider Context Provider acts as a module that produces new context information from internal or external information. The core capabilities of a Context Provider include capturing context information, data fusion, local caching, prediction, reasoning etc. In the same manner as Context Source, Context Provider exposes an interface for invocation and produces context information in a standardized message format.

Context Consumer Context Consumer retrieve contextual data through communication with context provisioning entities like Context Provider or even Context Source. Since the obtained context information can be processed, enriched, and made available for other consumer entities, a Context Consumer may act also as a Context Provider.

Context Broker Context Broker is the main component of each context-aware system with the important goal of providing a communication interface between other software entities in the overall system. The Context Broker has to register context provisioning components, perform their lookup, and discovery along with the control of context flow among all components.

Context Sensing

Capturing of context information is handled in the lowest layer which consists of a set of different sensors. Sensors extract raw data about the observed environment and provide an interface to access the information inherent in this data. It is worthy of note that the sensor concept includes every possible data source, which provides domain relevant information. According to the classification proposed in [28] sensors can be categorized into three groups:

- **Physical sensors** measure heterogeneous information about physical objects
- **Virtual sensors** extract information from virtual spaces like software application and services
- **Logical sensors** combine information provided by physical or virtual sensors to enrich the semantics of this information

In addition to this, sensors can be further classified into local and remote sensors. Since it is not possible for a single device to capture all context information as *local context*, the context management architecture should support connections to remote sensors to obtain context information managed by other entities (*remote context*) and by central brokers (*central context*) [12].

Context Modeling

In order to achieve a better processing of context data and provide interoperability between context information providers, Context Management standardizes a meta-model for context representation. An extensive and most cited survey of the most relevant context modeling approaches is provided in the paper by Strang et al. [29]. The analyzed approaches are classified according to the data scheme used to represent and exchange contextual information in the respective system.

Key-Value Models The most simple data structure is the key-value data model where data is represented in a form of a map between keys and values. Context data may be expressed as a collection of **name-value** pairs where **name** depicts the particular type of the context.

Markup Scheme Models Markup scheme modeling approach is based on the representation of information as a hierarchical data structure consisting of markup tags with attributes and content. A typical representative for a meta markup language is XML (*Extensible Markup Language*) which found wide acceptance and use in different fields of data representation and management.

Graphical Models modeling context information can be also achieved with general purpose graphical modeling instruments such as UML (*Unified modeling Language*) , ORM (*Object-Role Modeling*) or ER (*Entity Relationship modeling*) diagrams. Since these modeling tools provide extension mechanisms, contextual aspects can be implemented and made available as modeling tool extensions.

Object Oriented Models Applying object-oriented techniques to modeling of context knowledge brings benefits of Object Orientation such as encapsulation, reusability and abstraction. Different objects standing for various types of context encapsulate the details of context processing and provide access to contextual information through specified interfaces.

Logic Based Models Logic based models rely on the formal representation of contextual information and relations in terms of facts, expressions, and rules. Context information may be derived from a set of expressions or facts using logic conditions which are described in a set of rules in a formal logic based system. All logic based models are characterized by a high degree of formality.

Ontology Based Models Ontology based context modeling approaches inherit the strengths in the field of normalization and formality from ontologies. Representation of context knowledge with its semantics is achieved through conceptual ontology terminologies, formal axioms and constraints. Ontologies provide rich semantics embodied by a well-formed term vocabulary with clearly defined relations between different terms and inference rules used to derive new knowledge from existing facts [30]. In general, the ontological concept represent a promising context modeling approach because of its knowledge sharing, logic inferencing, and knowledge reuse capabilities [31]. Different techniques for ontology representation may be categorized into: *AI based*, *software engineering* (e.g. UML), *database engineering* (e.g. ER, EER) and *application oriented* (e.g. key-value pairs) techniques [12].

Machine Learning modeling The above mentioned classification of context modeling approaches is extended in the survey [32] by adding a new modeling concept based on *Machine Learning* techniques. Since categorization of context knowledge is a complex information processing task and ontologies fail to cover entire real-world context information, Machine Learning approach provides a better solution to model real-world context information. A Machine Learning method estimates the unknown mapping between system's inputs and its outputs from the available data, provides classification, association, and prediction of information as well as learning the model parameters in the presence of incomplete data.

In their extensive survey [5] Bolchini et al. present a general evaluation framework for analyzing context models with respect to a target application. The analysis framework is subsequently applied to the most interesting, data-oriented approaches found in the literature. The survey provides short description of the examined systems, highlighting relevant characteristics and the context modeling subproblems they are targeting.

Context Reasoning

To improve reusability of systems and ease application development, reasoning of context information is usually encapsulated, moved out of application logic and handled as a separate layer in the Context Management architecture. Depending on the specific purpose a context-aware system is designed for, the context reasoning layer may include the following features by adopting [5], [12] and [18]:

Context Construction As previously mentioned, context sensing is handled in the lowest layer consisting of a set of various sensors. Sensors simply provide the current contextual value which is a raw technical data retrieved from remote or local data sources. Hence, the context processing layer is responsible for transforming raw information using extraction operations and providing an access to a structured view of context information. Data structures presented in the context modeling layer incorporate the fundamental consideration to enable contextual data to be accessed in a context processing layer.

Ambiguity and Incompleteness Monitoring A context-aware system may encounter incomplete, incoherent, or even irrelevant context knowledge. The context reasoning layer provides mechanisms such as interpolation or mediation to deal with the processing of ambiguous data and the reconstruction of reasonable context information.

Context History and Storage Most context sources produce a continuous data stream which is relevant for all context-aware systems especially for those processing real-time scenarios. As sensors measurements may be valuable for later access and processing, the context reasoning layer addresses this issue by providing storage for historical context knowledge. Observing past measurements helps to derive knowledge about context, provide values during temporary sensor failures, and predict future context values.

Context Interpretation and Aggregation Only some of context-aware applications make use of raw context information retrieved directly from context sensing layer. Context consumers are mostly interested in already interpreted and aggregated data. Hence, the context reasoning layer is responsible for the transformation of raw, fine-grained data to a higher level abstraction and composition of information collected from several data sources to a high-level context information. According to the Context Toolkit architecture described in [13], the main components of context processing include *context interpretation* and *context aggregation* abstractions. Context interpretation defines the ability of the context reasoning layer to transform obtained context information including additional knowledge. Context aggregation defines a process which collects or aggregates context information from several sources and abstracts it onto a higher level. In [12], context aggregation happens in three ways:

- *One-to-One* transforms information obtained from one low level data source to one high level context dimension
- *Context Fusion* transforms information obtained from several low level data sources to one high level context dimension
- *Context Fission* transforms information obtained from one low level data source to several high level context dimension

Context Information Quality Monitoring To achieve high-quality data acquisition the context reasoning layer requires intelligent mechanisms for continuous quality testing of the retrieved context information and handling quality digression which is often caused by multiple aggregation of context.

Context Dissemination and Acquisition

The last and main component of the Context Management architecture involves dissemination of context information to interested applications. The key for distributing context knowledge is the architecture of context management systems which depends on various requirements such as system extensibility, resource availability, location and amount of context providers and consumers.

G. Chen et al. distinguish basically between two different approaches: *centralized* and *distributed* architectures [19]. A centralized architecture presents a common way to disseminate context information using a centralized context server which maintains all context information and provides access to information via standard API . On the other hand, a distributed architecture aims to store context knowledge in several places to avoid potential bottlenecks. It is also possible to think of a hybrid architecture where a centralized context server manages common context knowledge and each application maintains its own context data [12].

Another important issue is the kind of data acquisition and separation between managing and using context knowledge. In [33], H. Chen defines three categories of context acquisition methods:

- *Direct access to sensors*: Acquiring context information directly from sensors brings benefits such as control over operations of the low-level sensors and knowledge about context sensing and processing. According to this approach, context-aware applications are responsible for implementation of all context management layers including context sensing, modeling, and reasoning.
- *Middleware infrastructure*: The context sensing and modeling logic is encapsulated and handled outside the application logic. Using a middleware-based approach, context-aware applications focus on providing context reasoning logic.
- *Context server*: Shifting context acquisition logic to a centralized context server allows applications to acquire context and become context-aware without build-in context sensing capability. The context server implements all layers of Context Management architecture including context dissemination in a distributed environment.

The last important issue of acquiring and distributing context information is the communication style between context providers and consumers. Two different ways are widely used to gain access to context information: *synchronously* (pull-mechanism) and *asynchronously* (push-mechanism). In synchronous (pull) mode context context consumer sends a request to context provider and awaits response. This way, the context-aware application is responsible for polling the context server for context information changes. In asynchronous (push) mode, the context consumer subscribes for the context updates at the context provider. As soon as the required context information is available the context server either notifies about the update or pushes context data to the context-aware application.

To ensure robust and optimal management of a big amount of context information and its distribution over numerous context consumers in an efficient way, a single context server solution is often not feasible. *Context provisioning systems*, briefly discussed in the next chapter, offer a better context management solution.

2.2. Context Provisioning Systems

The main purpose of a context provisioning system is to provide an extensible and efficient approach for handling and distributing of context knowledge with respect to the general context management architecture outlined in the previous section. The context provisioning system integrates all the components required to maintain a high amount of distributed heterogeneous context sources, support alternative context modeling approaches, and provide extensive reasoning mechanisms for context knowledge. Furthermore, a context provisioning system is responsible for the dissemination of context information using basic synchronous and asynchronous context access patterns. This section briefly presents two relevant context provisioning platforms and describe their services.

2.2.1. Nexus Platform

Developed within the scope of the *Nexus* project [34] at the University of Stuttgart a context provisioning platform Nexus aims to provide support for all context-aware applications. The proposed solution relies on a novel idea to avoid development of context-aware applications using local context models tailored to best suit current application requirements. The Nexus approach suggests the integration of partial context models from a variety of context providers to build a shared global context model [35]. This so-called *Augmented World Model* builds upon the federation of compliant local context models called *Augmented Areas* by enabling the representation of context information using a global object-based ontology model with hierarchical *is-a* relationships between objects [36]. As depicted in Figure 2.3, the Nexus context provisioning platform consists of three different tiers:

Service Tier The Service tier abstraction maintains various context servers responsible for handling and providing access to Augmented Area information. To enable integration into the Nexus platform, each context server provides a standard interface to process request queries defined in the XML-based *Augmented World Query Language* (AWQL). AWQL provides standard data processing methods (insert, update, delete), as well as support for different spatial predicates (overlaps, inside, near). Furthermore, the context server returns resulting objects represented in *Augmented World Modeling Language* (AWML) to enable their further processing in the Federation tier.

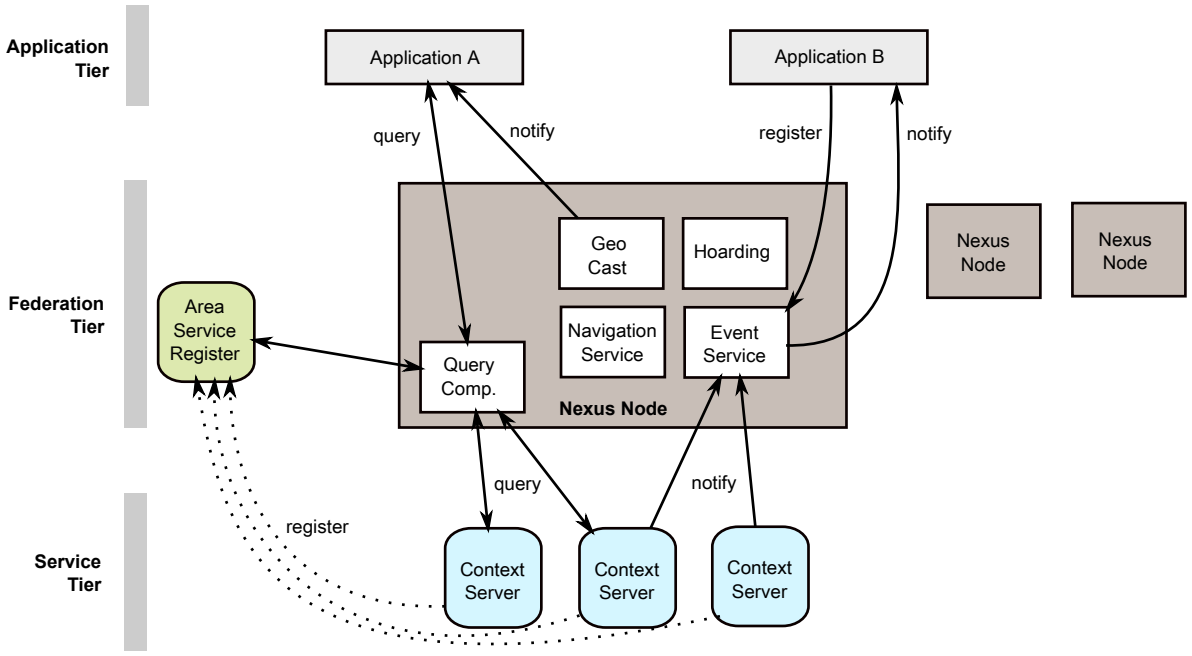

Figure 2.3.: Context Provisioning System – Nexus Platform [35]

Federation Tier Federation of different Augmented Areas into Augmented World Model is handled in this central tier. Mediating between the Application and Service tiers, the Federation abstraction is responsible for distributing request queries between available augmented areas, processing replies, and providing value added services such as monitoring of events and complex context aggregation. The *Area Service Register* (ASR), the main component of Federation tier, represents a directory for registering and discovering available context servers. The ASR stores information about integrated Augmented Areas, existing object types and their attributes, as well as the address of the server. It analyzes incoming request queries, detects and distributes query to the corresponding Augmented Areas and combines replies to provide a consistent view. In general, Nexus nodes provide the same interface as the context servers of the Service tier, support information caching and supply advanced value-added services mentioned above.

Application Tier The top tier provides a general abstraction for context-aware applications using the Nexus federated context model. The Nexus platform supports various interaction mechanisms such as processing request queries to retrieve context information, registering to the Event Service to forward notifications about context changes, as well as granting access to value-added services.

The Nexus platform as a context provisioning system provides extension mechanisms for integration of any desired context models, supports different communication paradigms, and allows implementation of various advanced value-added services on the platform basis.

2.2.2. C-CAST Context-Management Framework

C-CAST Context-Management Framework (C-CAST CMF) [37] is an ongoing EU-funded project which aims to provide an end-to-end context-aware communication framework for intelligent multicast-broadcast services. Among other important targets, C-CAST CMF addresses the key issue of service development for context representation, context assisted group management, and efficient context reasoning. This section briefly describes the C-CAST CMF architecture and gives an overview of the ContextML modeling language used in the framework. The detailed description will follow later in this thesis.

According to the general context management architecture outlined in the previous chapter, the context provisioning system C-CAST CMF consists of four main components: Context Consumer, Context Provider, Context Source, and Context Broker [38]. The Context Broker represents the core component of the entire architecture and is responsible for the communication between context provisioning and consuming entities. Context-aware applications generally referred to as Context Consumers leverage the services provided by Context Broker such as context aggregation, a Context Provider Lookup service, a Context Cache, and a Context History Service. Context provisioning entities like the Context Provider and Context Source provide context information in synchronous and asynchronous modes respectively. Context information and its exchange is generally represented through the concept of *entities* and *scopes*. Defined through a context type and a particular identifier, an Entity is basically a subject of interest where context data refers to. Scopes are designed to combine closely related context information into atomic units to enable consistent data operations such as creating, requesting, updating, and deleting.

To enable exchanging of context information between the architecture components, C-CAST CMF defines *ContextML* - a light weight XML-based language used for context representation and communication. Using ContextML for communication, Context Providers register themselves and their capabilities at the Context Broker. Context Consumers leverage ContextML for querying the Context Broker to find a particular Context Provider, obtain context information or formulate the event conditions, and receive event updates.

In this diploma thesis, the C-CAST CMF context provisioning system will be integrated into service compositions.

2.3. Integration of Context into Compositions

In this chapter different strategies to enable integration of context into service compositions are presented and evaluated. The objective of this survey is twofold. First, to give a brief description and identify the capabilities of each context integration strategy. The second goal is to analyze the existing integration techniques and to take this analysis as the starting point in order to define the requirements of the integration approach used in this thesis.

The remainder of this chapter is structured as follows: the first section presents a brief overview of the various context integration strategies. The second section introduces the evaluation criteria used for comparison of the integration approaches. Finally, the third section is devoted to the actual evaluation and draws conclusive considerations.

2.3.1. Solutions

Context Variables

George et al. present an architecture for modeling, sourcing, and propagating context information in WS-BPEL business processes based on the idea of *Context Variables* [24]. The basic concepts of their approach can be summarized as follows:

- Interpretation of context as a read-only environmental state external to a business process
- Standard-conform WS-BPEL language extension to support context variables
- Modeling of context sources as WS-Resources based on Web Service Resource Framework (WSRF)
- Propagating context changes from context sources to WS-BPEL variables using Web Service Notification (WSN) operations such as publish / subscribe

Their notion of context does not implicate any exact definition and categorization. Context is seen as a state external to a business process, whose value is not populated by the process and can change independent of a process' lifecycle, hence influence the process' execution. The representation of context information in WS-BPEL business processes is achieved through the language extensibility features of WS-BPEL by declaring *context variables*. Context variables are constructed and accessed inline in the same way as the standard WS-BPEL variables. The additional namespace-qualified attribute *isStateful=yes* is used to mark a context-aware business process as well as its variables storing context information.

Since a business process cannot change its context directly, the value of a context variable is populated by an external *publish / subscribe context source*. Context sources are defined to represent entities that expose a collection of environment state parameters or context. Modeling of context sources in web-services environment is handled outside the business process logic and achieved using standard WSRF constructs. This way, a context source is designed as a *WS-Resource*, which exposes a web-service interface for a business process to provide access to the underlying stateful resource. WSRF *WS-Resource factory* maintains the dynamic creation of WS-Resources at runtime and provides references to existing entities on demand.

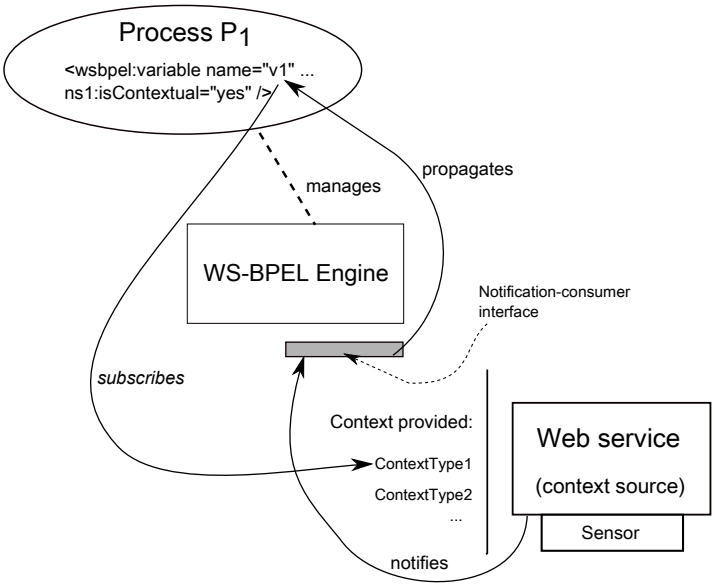

Figure 2.4.: Context Integration Approach – Context Variables [24]

WSN provides standardized *publish / subscribe* mechanisms for Web services where a WS-Resource acts as a notification producer and a WS-BPEL engine depicts a notification consumer (see Figure 2.4). In order to setup a business process' context variable, a workflow engine leverages the context type associated with a context variable and subscribes it to the corresponding context source. During the process execution WSN operations are used to continuously update context variables with recently changed context values from context sources.

The proposed concept of using context variables for context integration requires the modification of WS-BPEL engine. In particular, the WS-BPEL engine should implement logic necessary for interception and parsing responses for context parameters, linking context variables to available context sources, and updating context variables using WSN mechanisms.

Context Integration Processes

A different context integration approach is introduced in [39]. To meet the integration challenges Wieland et al. define an extensible modular architecture based on the *Integration Process Pattern*. As shown in Figure 2.5, a three layered system model provides the required abstraction to enable coupling of workflow engines with external information sources like context provisioning systems.

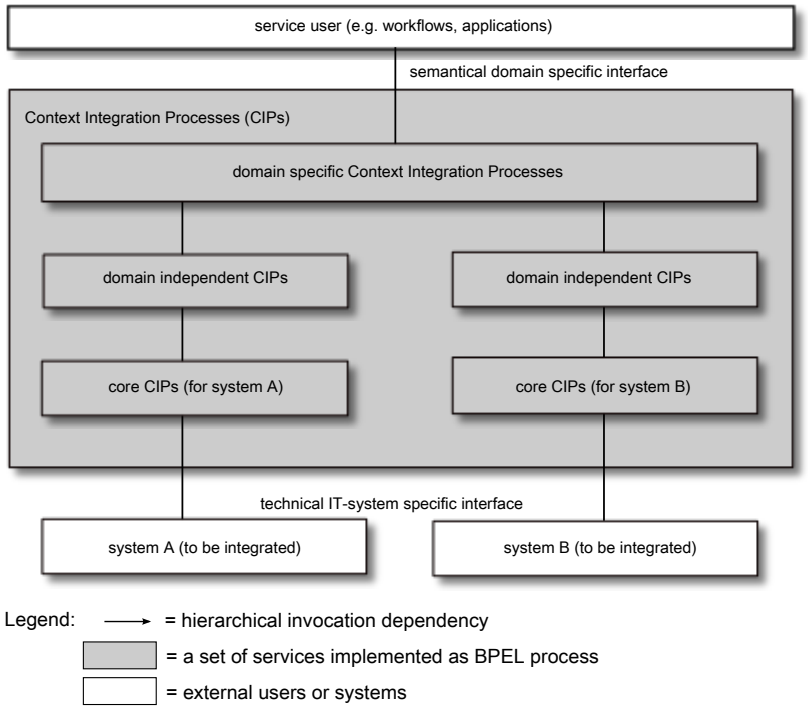

Figure 2.5.: Context Integration Approach – Context Integration Processes [39]

The *Smart Workflow Layer* realizes context-aware applications and in particular workflows, also called *Smart Workflows*. Implemented as standard WS-BPEL business processes the smart workflows represent technical processes that use context knowledge to be executed pervasively and adapt to changes in their physical environment.

Located at the bottom of the architecture the *Context Provisioning Layer* contains various context provisioning systems which represent the artifacts of the integration. Context provisioning systems provide generic services for managing context knowledge and distributing this information and functionality using synchronous and asynchronous communication patterns.

The *Context Integration Layer* mediates between the other two layers and represents the most important part of the architecture. The main responsibilities of this layer include the integration of the context provisioning components and the transformation between different message formats on the higher and lower layers to allow simplified access to the underlying systems' interfaces. To meet the outlined requirements the context integration layer is organized as a hierarchical Web service stack based on the Integration Process Pattern [40]. *Context Integration Processes* (CIPs) realize the hierarchical structure of the pattern and represent autonomous WS-BPEL workflows derived from each other. The CIPs are organized in three granularity levels:

- **Core CIPs** are responsible for the integration of the context provisioning components. Each service interface provided by a context provisioning entity is wrapped into one concrete CIP. If the interfaces of the integrated components change over time, only CIPs at the Core level should be modified accordingly. Thereafter, Core CIPs provide general, application independent context information, which allows the exchange of the underlying provisioning systems without changing the workflows at the Smart Workflow Layer.
- **Domain Independent CIPs** reduce interface complexity of the Core CIPs and can be reused in different application domains. The CIPs at this level may be used for finer grained selection of context data at different semantic levels.
- **Domain Specific CIPs** are tailored to the requirements of the particular application domain. The Smart Workflow Layer uses the services of the CIPs at this level to access Domain Independent CIPs in a simplified manner. Therefore, the Domain Specific CIPs are responsible for the transformation of the simple message formats on the application level to the more complex formats used at the Domain Independent level.

Propagating of context information in a Web services environment using the proposed approach is based on the existing standards and frameworks and does not implicate the modification of the WS-BPEL engine. Separating context integration into a hierarchy of different CIPs allows a high maintainability and reusability of the services. On the other hand, the numerous abstraction layers introduced in the Integration Process Pattern have a negative impact on the overall system performance.

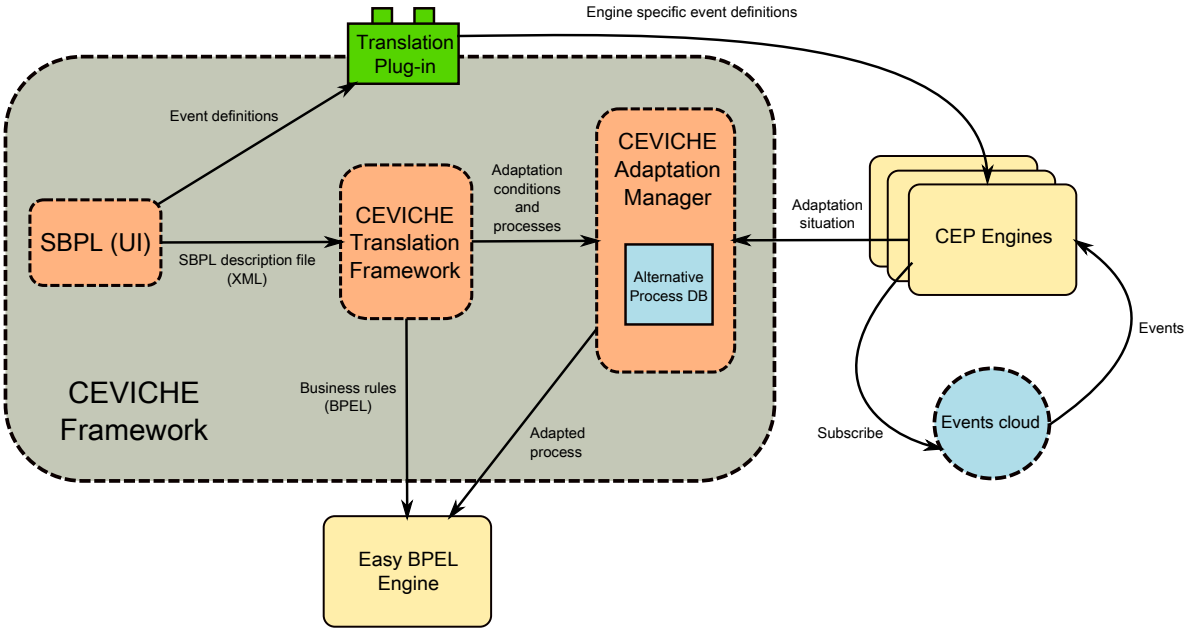

Figure 2.6.: Context Integration Approach – CEVICHE Framework [41]

CEVICHE

CEVICHE [41] stands for the ***C**omplex **EV**ent processI**n**g for **C**ontext-adaptive processes in pervasive and **H**eterogeneous **E**nvironments* and represents a framework which combines Complex Event Processing (*CEP*) and dynamic business process adaptation. According to the proposed approach, the context information is defined as various events which trigger the process adaptation during the execution. Hence, CEVICHE intends to facilitate the integration of different CEP engines using an plug-in approach.

CEVICHE relies on an extension of WS-BPEL which allows to extend business process' definition with adaptation points, conditions, and alternative processes in order to add flexibility to the process execution. The proposed *Standard Business Process Language* (SBPL) specifies the following additional elements:

- **adaptationPoint** is used to define the position in a process *where* the adaptation occurs.
- **situation** tag specifies the conditions expressed in the form of events indicating *when* the adaptation is expected.
- **adaptationType** defines *how* a process is adapted: before, after, or around the adaptation point.
- **alternativeProcess** is used to specify *what* is to be adapted in a business process.

A component-oriented approach enables CEVICHE to transform a process into dynamically bound components and define alternative processes to be dynamically integrated at runtime. The realization of this issue is achieved by using the *Easy BPEL* engine [42]. Easy BPEL represents process activities as independent components bounded according to the process definition. The bindings can be changed at runtime enabling adding and removing process components during the execution.

A CEP engine monitors the execution context by receiving and analyzing all the events generated by every change in the environment. To capture and extract relevant information a CEP engine relies on the previously defined CEP rules. These rules represent the conditions described in SBPL **situation** elements using XPath [43] expression language. CEVICHE supports different CEP engines by introducing a plug-in approach which allows to translate events and condition rules into the specified CEP engine's format.

As shown in Figure 2.6, the CEVICHE architecture features four components: 1. *SBPL User Interface* used for the definition of context-aware workflows, 2. *CEVICHE Translation Framework* that separates adaptation logic from the core business process, 3. *Translation Plug-in* responsible for translation of events and condition rules for each CEP engine, and 4. *CEVICHE Adaptation Manager* to enable process adaptation.

In order to use complex event processing for dynamic business process adaptation, an extended WS-BPEL engine is required. In addition, the proposed approach requires implicit knowledge in CEP technologies and available engines.

ContextServ

The *ContextServ* platform [44] has been developed within the scope of a research project sponsored by Australian Research Council (ARC) [45]. Adopting a model-driven development approach the ContextServ platform represents a generic approach for formalizing the design and development process of context-aware applications. ContextServ supports context provisioning management. Furthermore, it offers visual modeling tools for context and context-awareness techniques using a UML-based modeling language, called *ContextUML*. Finally, it provides a transformation tool for automatic generating of context-aware WS-BPEL processes out of the ContextUML model.

ContextServ supports the development of context-aware applications by adopting model-driven development concept. To enable modeling of context provisioning techniques, ContextServ introduces UML-based language, *ContextUML*. ContextUML provides high-level visual constructs for specifying the following context elements and context-awareness mechanisms:

Context modeling ContextUML defines two different types of context: *Atomic Context* and *Composite Context* standing for low-level and aggregated contexts respectively.

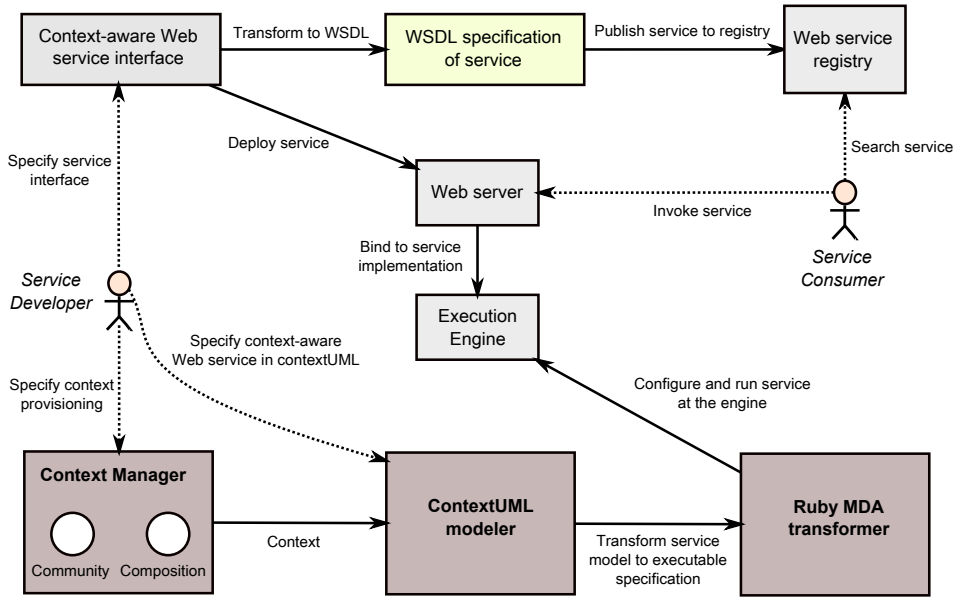

Figure 2.7.: Context Integration Approach – ContextServ Platform [44]

Context provisioning modeling The central modeling unit representing any context provisioning resources is *Context Source*. ContextUML distinguishes between two types of Context Sources: *Context Service* and *Context Service Community*. A Context Service represents a single context provider responsible for collecting, refining, and disseminating context information. At any time Context Services may enter and leave a Context Service Community that offers an unified interface to access aggregated Context Services. Since Context Services may distribute heterogeneous and ambiguous information, a Context Service Community is responsible for the dynamic provisioning of optimal context to the context consumer.

Context-Awareness modeling Context-awareness mechanisms are assigned to various *Context-aware Objects*. ContextUML adopts the structuring elements from WSDL (*Web Service Definition Language*) and defines the following context-aware objects: *Service*, *Operation*, *Message*, and *Part*. Two categories of context-aware mechanisms can be applied to these objects: *Context Binding* and *Context Triggering*. The former models the automatic binding of contexts to context-aware objects, like the mapping of an input parameter for a service object. The latter mechanism models the automatic contextual adaptation of context-aware objects using constraints and actions.

Using ContextUML visual editing and automation tools, context-aware applications are specified in a platform independent model and transformed into platform specific models such as WS-BPEL. As illustrated in the Figure 2.7, the ContextServ architecture contains the following main components:

- *Context Manager* provides functionality to specify context provisioning entities and register them within the ContextServ platform for future usage.
- *ContextUML Modeler* exposes a graphical interface for modeling of context-aware applications. The modeler is a version of ArgoUML [46] extended by adding a new diagram type ContextUML.
- *Ruby MDA transformer* is responsible for the transformation of ContextUML diagrams, exported in the XML Metadata Interchange (XMI) format, into an executable WS-BPEL context-aware process defined.

ContextServ realizes a rapid model-driven approach to develop context-aware Web services. This is the only solution, in this evaluation, that provides end-to-end support including visual editing tools. In addition, integration of context information into service compositions does not require any changes to the WS-BPEL execution engine.

2.3.2. Evaluation Criteria

The evaluation criteria listed below are intended to support the decision process, which context integration approach is best suited for future realization. The parameters used for future comparison are derived from the analyzed solutions, by selecting the most indicative and common ones. This section identifies the key issues and outlines the evaluation criteria as follows:

Context Modeling

The Context Modeling criteria evaluate the capabilities of the analyzed approach to address representation of context, its attributes, and context provisioning entities in service compositions:

- **Context Source** depicts the capability to address single context sources.
- **Integration of Context Provisioning Systems** defines if the analyzed approach supports large-scale context provisioning systems.
- **Context Aggregation** evaluates the capability of the underlying architecture to support aggregation and high-level reasoning of context information.
- **Context Quality** indicates if the integration approach explicitly ensures the quality of the retrieved context.
- **Technology / Techniques** defines how the context modeling is realized.

Context Awareness Modeling

The Context Awareness Modeling criteria evaluate which context-awareness mechanisms are available in the analyzed integration approaches.

- **Context Binding** defines if the context information is automatically bound to context-aware elements declared in a WS-BPEL process.
- **Context Triggering** indicates if context changes trigger the automatic execution of process operations.
- **Behavior Adaptation** depicts the capability of the approach to adapt business process logic at runtime.
- **Technology / Techniques** defines how the context awareness modeling is realized.

Service Composition

The Service Composition criteria evaluate the types of languages used for service composition and their modifications.

- **WS-BPEL standard language** indicates that service composition supports standard WS-BPEL.
- **WS-BPEL language and engine extension** states that service composition is based on an extension of the WS-BPEL standard.
- **Support for other composition languages** evaluates if the integration approach can be applied for other service composition languages than WS-BPEL.

Complexity

Complexity criteria examines the realization of the available context integration solutions based on their complexity. The realization difficulty is rated according to the scale: *Hard*, *Challenging*, and *Easy*.

- **Context Modeling** denotes the complexity when realizing the issues mentioned under the Context Modeling criteria such as context representation and context provisioning.
- **Context Awareness Modeling** denotes the complexity when realizing the mechanisms outlined under the Context Awareness criteria.
- **Integration of Context into Service Compositions** denotes the complexity when realizing the processing of the context information at the business process logic level.

- **Portability** denotes the complexity when realizing the exchange of underlying infrastructure for context-aware processes.

Modeling Support

The Modeling Support criteria evaluate if the examined context integration approaches provide support for visual modeling of context, context awareness, and service composition.

2.3.3. Comparison

This section analyzes and compares available context integration solutions with respect to the previously outlined comparison criteria. The evaluation result provides a structured view of the state of the art, which enables focusing the attention on the key issues of context integration. Finally, the best context integration approach is selected.

The capabilities of each context integration strategy are identified in the Tables 2.2 and 2.3, where the rows depict various evaluation criteria along with the corresponding key issues and columns represent the discussed context integration solutions. The supported features are denoted with the sign *X*. In cases where no clear statement is given whether the capability is supported by the particular integration approach, the feature field is left blank.

Categories	Capabilities	Context Variables	Context Integration Processes	CEVICHE	ContextServ
General	**Context Definition**	Context is an environmental state which is external to a process, whose value can change independent of process' life cycle, and can influence the process' execution. A process cannot directly change its context and only its designers can decide what qualifies as context in their applications.	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 (based on Dey and Abowd, 1999).	Context is defined as events which trigger the business process adaptation.	A context-aware Web service is a Web service that uses context information to provide relevant information or services to users.
Context Modeling	**Context Source**	✓	✓	–	✓
	Integration of Context Provisioning Systems	✓	✓	✓	✓
	Context Aggregation	–	✓	✓	✓
	Context Quality	–	–	–	✓
	Technology / Techniques	WSRF	WS-BPEL	Complex Event Processing	ContextUML Model Driven Development
Context Awareness Modeling	**Context Binding**	✓	✓	✓	✓
	Context Triggering	✓	✓	✓	✓
	Behavior Adaptation	–	–	✓	–
	Technology / Techniques	WSN	WS-BPEL	Complex Event Processing, Component Oriented Programming	ContextUML Model Driven Development
Service Composition	**Static**	✓	✓	✓	✓
	Standard WS-BPEL	–	✓	–	✓
	WS-BPEL language and engine extension	✓	–	✓	–
	Support for other composition languages	–	–	–	✓

Table 2.2.: Comparison between Various Strategies of Context Integration into Service Compositions (Part 1)

Categories	Capabilities	Context Variables	Context Integration Processes	CEVICHE	ContextServ
Complexity	**Context Modeling**	**Hard** - Full context schema modeling - Context Source implementation in WSRF	**Challenging** Integration or new implementation of context provisioning system.	**Hard** - Integration of CEP engine - Definition of CEP rules to generate events	**Challenging** ContextUML provides mechanisms to model context types, integrate context sources and aggregate them as context communities. ContextUML defines Quality of Context parameters to provide dynamic context source selection.
	Context Awareness Modeling	**Challenging** - Publish-Subscribe configuration with WSN - Automatic context binding and triggering	**Hard** Implementation of Context Integration Processes.	**Challenging** Specification of adaptation points in BPEL and adaptation logic including alternative business processes for adaptation.	**Challenging** Modeling of business process with UML-based language ContextUML is challenging. Context Awareness is modeled at this abstract level which is then automatically transformed into BPEL.
	Integration of Context into Service Compositions.	**Easy** Marking of BPEL variables with `isContextual` tag.	**Easy** Invocation of services of Domain Specific Processes.	**Easy** Identification of adaptation points in business processes and binding of specific events to these adaptation points.	**Challenging** Modeling of business process with UML-based language ContextUML is challenging. Context information is integrated into business processes at this abstract level.
	Portability	**Hard** Composition engine should support: - extended version of WS-BPEL - WSN and WSRF	**Easy** Composition engine should support standard WS-BPEL.	**Hard** Composition engine should support: - business process modification at runtime - extended version of WS-BPEL	**Easy** Composition engine should support standard WS-BPEL.
Modeling Support	**Context**	–	–	–	✓
	Context Awareness	–	–	–	✓
	Service composition	–	–	–	✓

Table 2.3.: Comparison between Various Strategies of Context Integration into Service Compositions (Part 2)

According to the main requirements outlined in the current thesis statement, the selected context integration approach should be based on the non-extended WS-BPEL standard and utilize standard compliant techniques for modeling of context and context-awareness mechanisms. Further requirements include the integration of large-scale context context provisioning systems and support of high level context aggregation. Finally, the chosen context integration strategy should provide the capability to graphically model the context integration into service compositions.

The evaluation presented in Tables 2.2 and 2.3 points to *Context Integration Processes* as the ideal approach for integration of context information into service compositions. This conclusion represents a specific view and depends on the emphasized integration requirements.

Chapter 3

Integration of Context Information

The previous chapter was devoted to provide a fundamental background knowledge necessary to understand the concepts and principles of context-aware applications. Various context provisioning systems and their integration into compositions were presented, discussed and compared regarding their complexity. This chapter describes the realization of the C-CAST CMF integration into service compositions using Integration Process Pattern as the best suited integration strategy according to the previous evaluation. The first section describes the main components of the C-CAST Context Management Framework (CMF) architecture. The detailed specification of currently supported component interfaces, protocols, and message formats is available in the Appendix Sections A.1 and A.2 on page 91, and A.3 on page 98 of this diploma thesis. The overview is followed by a proposed integration architecture based on the Integration Process Architecture Pattern.

3.1. C-CAST CMF

C-CAST CMF context provisioning system was briefly introduced in the Chapter 2.2.2 on page 22. This section is focused to describe the core components of the framework and identify the integration artifacts and exposed service interfaces.

3.1.1. Architecture Components

The context provisioning system C-CAST CMF consists of four main components: Context Consumer, Context Provider, Context Source, and Context Broker [38]. Figure 3.1 illustrates the basic components of the framework.

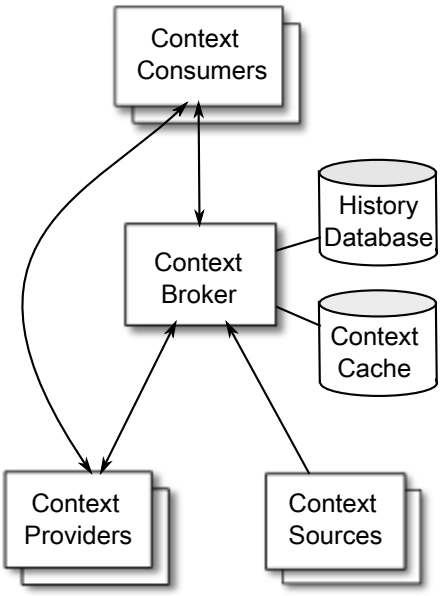

Figure 3.1.: C-CAST CMF – Architecture Components

Context Broker This component acts as a central module of the entire C-CAST CMF architecture. The Context Broker facilitates and controls the communication between the components providing and consuming context information. Available context provisioning entities use announcement process to register their capabilities and communication endpoints to the Context Provider Lookup Service maintained by the Context Broker. Additionally, the Context Broker supports both synchronous and asynchronous communication mechanisms to distribute and acquire context knowledge, provides necessary caching of context information, as well as context history service.

Context Providers The basic context provisioning entities are responsible for the collection of context information from a set of sensors, network, services or other data sources and synchronous dissemination of the obtained context knowledge to the numerous context consumers. Each Context Provider may also support reasoning of high level context information using various interpretation, filtering, and aggregation mechanisms.

Context Sources As a special case of context provisioning entities, Context Sources provide context information in an asynchronous mode. Hence, they do not expose any interfaces to query context information directly. Without being queried Context Sources utilize asynchronous push mechanisms to notify Context Consumers about available updates or send updated context knowledge to the Context Broker and other Context Consumers. The context information is stored in the broker's Context Cache and linked to an expiration timer.

Context Consumers The architecture components that acquire and use information from context provisioning components are named Context Consumers. In order to gain synchronous access to the context data Context Consumers query Context Broker or invoke particular Context Provider directly. In case the information should be retrieved asynchronously, the Context Consumers subscribe for context knowledge of a certain type at the Context Broker which deliver notifications about updates and changes. Furthermore, Context Consumers leverage various services provided by the Context Broker such as a Context Provider Lookup service, a Context Cache, and a Context History Service.

3.1.2. Context Broker Services

Context Caching In order to deal with the unnecessary performance overhead caused by the repetitious acquisition and initialization of the same information resources, the Context Broker stores the obtained context information in a Context Cache. The expiration mechanism is applied to maintain the validity of the context information and remove stale information. Additionally, the information retrieved from the Context Sources is automatically stored in the Context Cache, since Context Sources do not expose interfaces to query context data directly.

Context Validity The validity of context information is stated through the creation and expiration timestamp. The expiration time is defined by the Context Source or Context Provider responsible for provisioning of the current context data. The Context Broker may in turn change this expiration time information to synchronize it to its internal clock.

Context History The Context History service utilizes logging of the context information exchanged between context provisioning and consuming entities to enable later access and reasoning of stored knowledge. Observing past measurements helps to derive knowledge about context, provide values during temporary sensor failures, and deduce further context information, for example, about situations and user intentions.

3.1.3. ContextML Model

To facilitate communication between the architecture components C-CAST CMF defines *ContextML* – a light weight XML-based language used for context representation and communication with C-CAST CMF. Using ContextML for communication, Context Providers register themselves and their capabilities by the Context Broker. Context Consumers leverage ContextML for querying the Context Broker to find a particular Context Provider, obtain context information or formulate the event conditions, and receive event updates. To present the core elements of ContextML, this section outlines the fundamental concepts used in CMF such as the *Entity* and *Scope* concepts (see Figure 3.2).

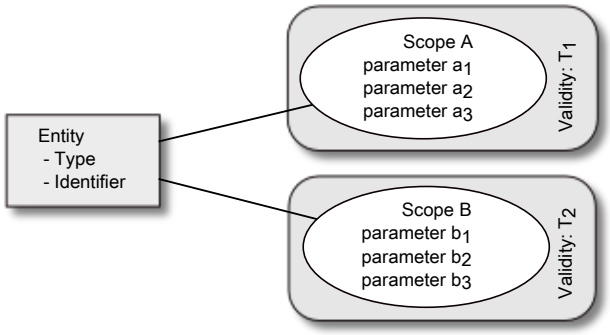

Figure 3.2.: C-CAST CMF – Entity Scope Relationship [38]

Entities The C-CAST context management framework relies on the *Entity* concept to enable mapping of exchanged context information to specific objects. An Entity represents a subject of interest which can be in turn a complex group of multiple entities. Two elements are considered to be sufficient to model entities: *entity type* and *entity identifier*. Whilst an entity type is used to categorize a set of entities, an entity identifier specifies a particular item in a corresponding set of entities. Names of entity types and entity identifiers are case sensitive, so that different use of uppercase and lowercase letters lead to different entity names.

The following represent sample entity tuple names:

- `username | johnd`
- `imei | 123456`
- `cell | 01761234567`

Listing A.1 on page 92 included in the Appendix Section A.1 of this diploma thesis demonstrates entities supported in the current C-CAST CMF version.

Scopes The context information exchanged between components of the CMF architecture always relates to a particular entity and is represented as a set of closely related context parameters which are logically organized in so-called *Scopes*. Identified with a unique name, scopes are used as a primary context exchange unit to obtain the consistent view of exchanged context information. Contained context knowledge is wrapped into corresponding context parameters which belong only to one particular scope. A scope can be atomic or contain a union of different atomic scopes. The Scope approach ensures the atomicity properties of all operations on exchanged context data. A sample entity-scope association is illustrated in Figure 3.2. A detailed specification of the context scopes defined and supported in the current version of the C-CAST CMF is included in the Appendix Section A.2 of this diploma thesis.

3.1.4. RESTful Web Service Interface

The C-CAST context management framework exposes RESTful Web service interface to enable the integration with other context-aware applications and provide access to context information and other value-added services. This section is devoted to the description of the exposed interfaces and exchanged message formats supported in the current C-CAST CMF version. Available CMF service interfaces are organized according to the main components of the outlined CMF architecture.

Context Broker Interfaces A Context Broker request should be of the following form:

`HTTP://<HOSTNAME>:<PORT>/CB/ContextBroker/method?parameters`
Where `method` may be either of the following values:

- `getName` returns general information about current Context Broker version
- `providerAdvertising` is used to plug in a new Context Provider
- `getContextProviders` returns information about registered Context Providers
- `getActiveEntities` returns information about active entities
- `getContext` is used to retrieve specific context information
- `contextUpdate` is used to create, update, or delete context information

The detailed description of each invocation method along with the corresponding parameters is included in the Appendix Section A.3 on page 98 of this diploma thesis. Where possible, the input and output messages are described using W3C XML schema.

Context Provider Interfaces A Context Provider request should be of the following form:

`HTTP://<HOSTNAME>:<PORT>/<CP_MODULE>/<CP_INTERFACE>/method?parameters`

Where `CP_MODULE` and `CP_INTERFACE` represents a Context Provider module and its corresponding interface respectively. A `method` may be either of the following values:

- `getContext` is used to retrieve specific context information
- `providerMethod` stands for a specific method advertised by a particular Context Provider

The Appendix Section A.3 on page 98 includes the description of the methods exposed by Context Providers. The context query interface of a Context Provider has the same syntax and semantics as the related interface exposed by the Context Broker. Additionally, the implementation of a specific provider method varies from every other registered Context Provider. Hence, the detailed specification of a Context Provider API should be furnished along with its implementation.

3.2. Challenges

The general concept of Context Integration Processes was introduced in the section devoted to different context integration solutions (see Section 2.3.1 on page 25). This section summarizes the properties of the adopted integration concept and covers the challenges which occur during the realization of context integration processes.

3.2.1. Concept Properties

Based on the Integration Process Architecture Pattern, the Context Integration Processes represent a three layered system model realized as a hierarchical Web service stack. Using Web services to define integration processes introduce a set of useful advantages:

System Abstraction A loosely coupled architecture enables integration of context provisioning systems to be more manageable. Interfaces of the integrated context provisioning system may evolve over time without forcing context-aware applications to adapt their business logic to the changes. Even the underlying context provisioning systems may be easily exchanged, since this approach provides the necessary system abstraction.

Simplified Access Interface Context integration processes reduce the interface complexity of the context provisioning components. Domain Specific CIPs of the higher level are tailored to the requirements of the specific application domain and provide simplified access and finer grained selection of context information.

Reusable Components An extensible modular architecture enables easy reuse of existing context integration processes and their incorporation into different application domains. On demand, new context integration processes of different semantic levels can be built upon existing components and be exposed to context-aware applications.

Context Reasoning and Aggregation To simplify context-aware application development, reasoning and aggregation of context information can be encapsulated, moved out of the business logic, and handled as separate context integration processes. A sample aggregation of context information takes place if context knowledge originates from different context provisioning entities or various context consumers subscribe to the same context updates.

3.2.2. Integration Challenges

Since C-CAST CMF is an ongoing development project, new versions will evolve with the time. The successful integration of C-CAST context management framework introduce a set of challenges which should be addressed in the current realization.

ContextML Data Model Evolution The modification of the underlying ContextML XML Schema data model should not implicate the adaptation of context-aware applications. Core and Domain Independent CIPs should encapsulate the interfaces of the C-CAST CMF services and support forwarding of communication messages wrapped in a `contextML` root element to the integration processes of the higher level. The adaptation to the changes of ContextML data model will happen at the level of Domain Specific CIPs, since this system layer is responsible for acquiring application specific context information and possible context reasoning.

Service Interface Modification In a similar way, modifications of C-CAST CMF service interfaces should not implicate the adaptation of context-aware applications. Core and Domain Independent CIPs are responsible for the adaptation to the interface modification of C-CAST CMF services.

Deployment of Additional C-CAST CMF Components and Services Additional C-CAST CMF components and services should be easily deployed using existing or by incorporating new Core and Domain Independent context integration processes.

3.3. Realization of Context Integration Processes

This section presents the realization of context integration processes to meet the requirements outlined in the previous sections.

3.3.1. Context Provisioning Layer

Located at the bottom of the architecture, the Context Provisioning Layer contains the services provided by the various systems which represent the artifacts of integration. Context provisioning systems should provide appropriate interfaces to their services in order to facilitate their integration using WS-BPEL.

According to the WS-BPEL specification [2], the WS-BPEL standard is tightly coupled with the service model defined by Web Service Description Language (WSDL) [47] and implies that both the business process and its partners are exposed as WSDL services. However, the RESTful Web service API of the C-CAST context management framework does not rely on the standard WSDL to describe available interfaces of the system. Hence, it is not possible to directly integrate the services of the current context provisioning system using the non-extended WS-BPEL standard. One of the important requirements outlined in this thesis states that current integration solution should utilize WS-BPEL standard to be deployed across multiple composition engines. To address this requirement, the RESTful API of the C-CAST context management framework will be wrapped into standard Web service interface exposed through WSDL and accessed via SOAP over HTTP. This approach will obviously reduce the performance of the overall system. But since this is a temporary solution and the C-CAST context management framework still evolves, it might be expected that future version of C-CAST CMF will expose appropriate WSDL interface.

The Java API for XML Web services (JAX-WS 2.0 [48]) was selected for the realization of the SOAP Web service interface for the C-CAST context management framework. In particular, Apache CXF [49] was taken as the most suited JAX-WS reference implementation according to the performance comparison provided in [50]. Apache CXF also supports Java API for RESTful Web services (JAX-RS [51]) to develop HTTP centric clients to communicate with RESTful services. The version of Apache CXF used for implementation was 2.x. The development of the CMF SOAP Web service interface was realized in the Integrated Development Environment Eclipse [52] version 3.6.2 with Java 1.6. Finally, the SOAP Web service interface for the C-CAST context management framework was deployed as a Web application to Apache Tomcat version 7.0, an open source servlet container [53].

According to the overview over the interfaces of the C-CAST CMF architecture components provided in the previous section, the central component Context Broker appears to be the main artifact of the integration. This paragraph summarizes the Context Broker services selected for integration. As shown in Figure 3.3, the integrated services include:

- `GetBrokerContext` service
- `GetActiveEntities` service
- `ContextUpdate` service
- `GetContextProviders` service

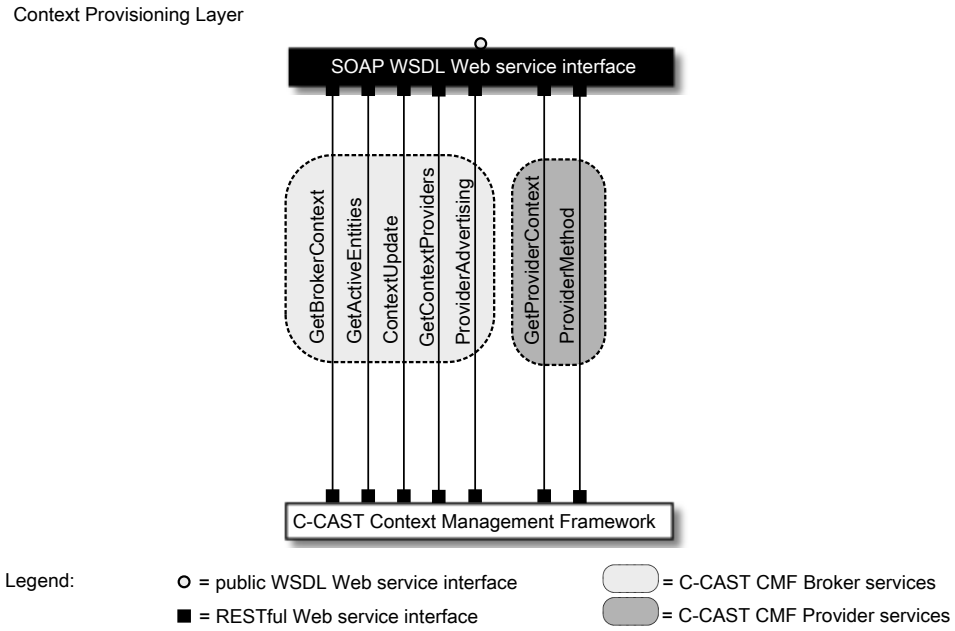

Figure 3.3.: C-CAST CMF – SOAP WSDL Web Service Interface

- `ProviderAdvertising` service

Additionally, a generic interface for a Context Provider was implemented. The SOAP Web service interface of this component include:

- `GetProviderContext` service
- `ProviderMethod` service

The `ProviderMethod` service is implemented to be generic. This service expects as an input parameter an additional information about the communication endpoint to invoke a specific provider method. Other input parameters passed to the RESTful Web service interface are represented as a map of *string* values specifying name and value of a particular parameter.

Listing 3.1 CXF Default SOAP Fault Response

```
<soap:Fault>
  <faultcode>soap:Server</faultcode>
  <faultstring>cxf fault</faultstring>
  <detail>
    <contextML xmlns="http://ContextML/1.7" ...>
    <ctxResp>
    ...
    <resp code="455" msg="Context Element Not Available: No Context Available to Invoke
        Provider LP" status="ERROR"/>
    </ctxResp>
    </contextML>
  </detail>
</soap:Fault>
```

To facilitate fault handling in upper layers of the context integration processes each C-CAST CMF service returns a fault message in case of invocation failures and other errors. Generated fault messages contain error code, response status, and ContextML element describing the actual cause of the invocation failure.

It is worthy noting, that while generating SOAP Fault messages the CXF framework sets the default value for SOAP `faultcode` to `soap:Server` as shown Listing 3.1. Since BPEL `bpel:faultHandlers` leverage the fault code value to identify the corresponding `bpel:catch` clause, the fault code value should contain the real name of the thrown fault. In order to modify the SOAP Fault message generated by the CXF framework, an *outbound fault interceptor* was defined and added to the CXF fault chain. Each message received or sent by the CXF framework go through a series of so-called inbound and outbound interceptors. The last interceptor in the outbound fault chain is *Soap11FaultOutInterceptor* that appends the fault code and details information into the response. Hence, a new interceptor was implemented as the extension to CXF *AbstractSoapInterceptor* and subsequently added to the outbound fault chain. Listing 3.2 illustrates the definition of the new interceptor in the CXF configuration file.

Top down development was chosen for the realization of the Web service proxy server for C-CAST CMF services. The starting point was to specify the C-CAST CMF service interfaces using WSDL and define request message formats using XML schema. Separate WSDL interface specifications were developed for Context Broker and Context Provider services. The next step was to generate the server side code using the *WSDL to Java* tool provided by Eclipse. Finally, each method representing a separate C-CAST CMF service was implemented as a HTTP client to communicate with the RESTful interface.

Listing 3.2 Registering Fault Interceptor in CXF Service Configuration

```
<jaxws:endpoint xmlns:tns="http://www.ict-ccast.eu/CMF/ContextBroker/definitions"
  id="contextbroker"
      implementor="eu.ict_ccast.cmf.contextbroker.definitions.ContextBrokerImpl"
  wsdlLocation="ContextBroker.wsdl" endpointName="tns:ContextBrokerSOAP"
  serviceName="tns:ContextBroker" address="/ContextBrokerSOAP">
  <jaxws:features>
    <bean class="org.apache.cxf.feature.LoggingFeature" />
  </jaxws:features>
  <jaxws:outFaultInterceptors>
    <bean class="eu.ict_ccast.cmf.contextbroker.definitions.FaultOutInterceptor" />
  </jaxws:outFaultInterceptors>
</jaxws:endpoint>
```

3.3.2. Core Integration Processes

The main responsibilities of the context integration processes of this layer include encapsulating each particular service interface of the context provisioning system into one concrete WS-BPEL business process. The context integration processes are named according to the C-CAST CMF services they represent. Hence, the selected names describe their functionality. The integration processes at this level encapsulate all services currently provided by the C-CAST context management framework and illustrated in Figure 3.4.

Context Broker processes:

- *GetBrokerContext* integrates the querying functionality of the C-CAST context management framework. To obtain context data a list of entities and corresponding context scopes should be provided. The ContextML message format is used for the query result presentation.
- *GetActiveEntities* is used to obtain information about the active entities. The result of the operation is presented in the ContextML format.
- *ContextUpdate* integrates the context manipulation functionality. The ContextML message format is used for the context manipulation request and result presentation.
- *GetContextProviders* is used to acquire information about registered context providers. The result of the operation is presented in the ContextML message format.
- *ProviderAdvertising* integrates the functionality of C-CAST CMF to advertise and register a new Context Provider or Context Source. *ProviderAdvertising* integrates the ContextML message format for the advertisement and registration result presentation.

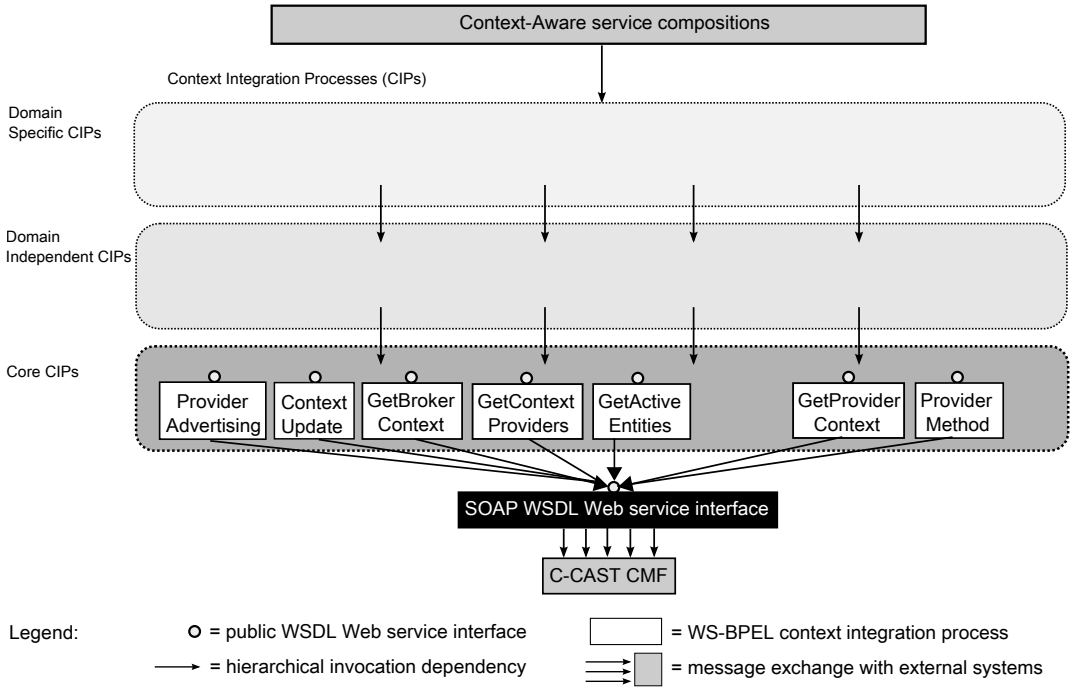

Figure 3.4.: C-CAST CMF – Core Context Integration Processes

Context Provider processes:

- *GetProviderContext* has the same semantics as the related Context Broker business process integrating the querying functionality of the Context Provider. The context information queries contain information about entities and context scopes. Query results are declared using the ContextML message format.
- *ProviderMethod* encapsulates a specific Context Provider service. The business process expects information about Context Provider communication endpoint as an input parameter. Other parameters passed to the *ProviderMethod* are represented as a map of *string* values specifying name and value of a particular input parameter to a RESTful Web service.

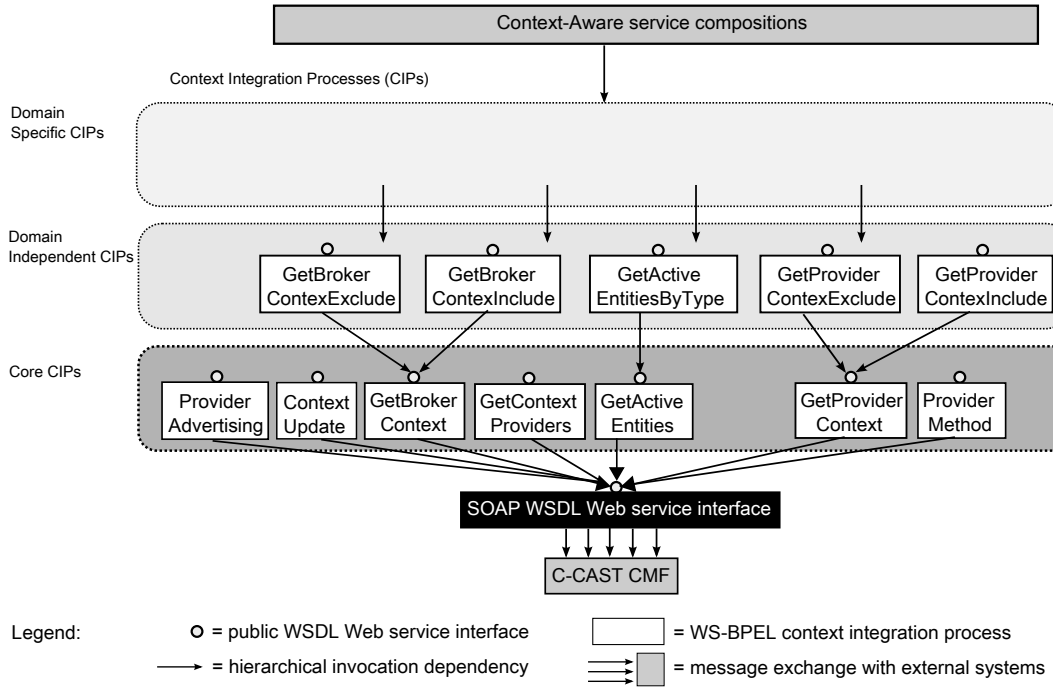

Figure 3.5.: C-CAST CMF – Domain Independent Context Integration Processes

3.3.3. Domain Independent Processes

Context integration processes at this layer basically reduce interface complexity of the Core CIPs. Figure 3.5 illustrates the Domain Independent CIPs defined and implemented to integrate C-CAST CMF.

Context Broker processes:

- *GetBrokerContextExclude* is a specialization of the Core CIP *GetBrokerContext* used to minimize the size of the result set. *Extensible Stylesheet Language Transformations* (XSLT) [54] stylesheets are defined for matching the result values against a set of the input attributes and blanking out the returned elements which contain specific context information.
- *GetBrokerContextInclude* is another specialization of the Core CIP *GetBrokerContext* used for blanking out the result elements which do not contain specific context information. *XSLT* stylesheets are defined for matching the result values against a set of the input attributes.
- *GetActiveEntitiesByType* is used to acquire a list of the active entities of a specific type. This CIP is based on the Core CIP *GetActiveEntities*.

Context Provider processes:

- *GetProviderContextExclude* is a specialization of the Core CIP *GetContext* used to minimize the size of the result set. GetProviderContextExclude is realized in the same way as the related CIP of the Context Broker. This CIP uses the Core CIP GetContext to query context information.
- *GetProviderContextInclude* is another specialization of the Core CIP *GetContext* used for blanking out the result elements which do not contain specific context information. *GetProviderContextInclude* is realized in the same way as the related CIP of the Context Broker. This CIP uses the Core CIP *GetProviderContext* to query context information.

3.3.4. Domain Specific Processes

Since context integration processes at this layer are specific to the particular application domain, this section does not describe any of them. Sample Domain Specific processes tailored to the requirements of the example application will be defined in the Evaluation Chapter 5 on page 65 of this diploma thesis.

Chapter 4

Aggregation of Context Information

In this chapter, the preliminary considerations related to the aggregation of context information are introduced and subsequently boiled down to a generic aggregation approach. The starting point is the development of different aggregation scenarios to give a general idea of how aggregation of context information is done. The next step is devoted to the identification of the aggregation process components and their integration into the Integration Process Architecture Pattern. Following the detailed description Google Maps Web services are integrated using Context Integration Processes to demonstrate how the proposed concept can be applied to aggregate services provided by C-CAST CMF and Google Maps Web services.

4.1. Definition of Information Aggregation

Information aggregation represents a process intermediating between data provisioning and consuming entities which handles heterogenous information obtained from multiple data sources to provide value-added information view for multiple data consumers. The goal of information aggregation is, given various information sources, to extract important information, process, and present the content in a format and manner suitable to the consumer requirements. A number of different expressions represent the activity of aggregation: *extract*, *abstract*, *compare*, *transform*, *merge*, *reduce*, *refine*, and so on. Basically, the aggregation logic is tailored to best suit current application requirements. In general, the aggregation techniques can be grouped as follows:

Identifying and Mapping Relevant Information The basic responsibility of any aggregation process is to obtain data from available provisioning sources and identify information relevant to the application domain. Additionally, an aggregation process provides the mapping between data sources and data consumers. It disseminates acquired information to various information consumers according to their needs. Hence, an aggregation process enables information consuming entities to obtain a specific data view on multiple and heterogeneous data sources.

Transforming the Information Structure Once the important contents from the original information are extracted, an aggregation process may abstract data representation to obtain a common view on heterogeneous data types. If necessary, the obtained information may be reduced, refined, structured, and organized in a specific order.

Processing the Information The general idea behind the information aggregation is to process the obtained information in order to address specific application requirements. Being retrieved from various data sources information might be merged, compared, or used to generate other valuable information.

4.2. Information Aggregation in Service Compositions

Common properties shared among aggregation processes can be identified with the goal to define generic aggregation process patterns. These patterns can be subsequently used to simplify development of new context aggregation processes.

Most of the aggregation processes which do not perform complex mathematical aggregation logic can be modeled using the WS-BPEL standard. WS-BPEL utilizes several XML specifications (XML Schema 1.0 [55], XPath 1.0 [43] and XSLT 1.0 [54]) and provides comprehensive support for data manipulation. XPath expressions can be used to access the values of process variables. Data transformation and processing can be accomplished using XSLT and WS-BPEL standard processing mechanisms.

Since aggregation logic is tailored to the specific requirements of each particular application, it is not possible to provide generic templates to support various aggregation techniques outlined in the previous section. Though XPath and XSLT offer a full toolkit for exploiting custom application logic, a full understanding of the process data structures and control flow is required to enable specific data transformation and processing. Hence, the proposed aggregation approach does not define any variant templates for data transformation and processing.

Instead of trying to discover generic templates to support custom data transformation logic, we will concentrate on the facilities provided by WS-BPEL and try to discover how these facilities can be used to express similarities between aggregation processes. Primarily, WS-BPEL is used to model interactions between Web services and assemble external entities defined using WSDL into an end-to-end process flow. Since aggregation processes combine various data sources, their cumulative similarities rely on the communication style with the external entities and number of involved data sourcing and consuming parties.

Number of Data Sourcing and Consuming Entities As shown in Figure 4.1, the interaction between external entities and the aggregation process happens in various ways by adopting observations provided in [12].

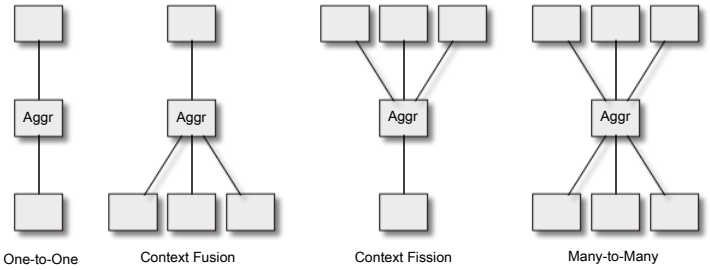

Figure 4.1.: Aggregation – Number of Data Sourcing and Consuming Entities

- *One-to-One* transforms information obtained from one low level data source to one data consuming entity of higher level.
- *Context Fusion* transforms information obtained from several low level data sources to one data consuming entity of higher level.
- *Context Fission* transforms information obtained from one low level data source to several data consuming entities of higher level.
- *Many-to-Many* transforms information obtained from several low level data sources to several data consuming entities of higher level.

Communication Styles Communication styles represent another important similarity aspect between aggregation processes. Each aggregation process obtains and disseminates the context information using one of the following patterns of communication:

- *Pull mode* represents an interaction style where the communication request originates at the client side and the server generates a response message. WS-BPEL defines the `<invoke>` activity to call Web service operations provided by service providers. According to the WSDL specification, operations are classified into synchronous (request / response operations) and asynchronous (one-way operations).

- *Push mode* defines an opposite interaction style, where the server side initiates the communication by pushing messages to the clients. WS-BPEL supports push mode communication by introducing at the server side one-way `<invoke>` activities to push information to the consuming entities and `<EventHandlers>`, `<receive>`, and `<pick>` at the client side to react when specific events occurs.

The application of communication styles to various data provisioning and consuming entities may be used to express various aggregation processes. Based on this experience a command-line tool can be developed to support automatic generation of aggregation process skeletons. This will allow designers 1. to simplify binding definition to external services, 2. to define the overall structure of an aggregation process including logic blocks describing the interaction with external services, and 3. to implement their own data transformation logic. To generate an aggregation process, designers specify data provisioning and consuming entities exposed as Web services and provide necessary arguments such as:

- Number of data provisioning and consuming entities
- WSDL interface specifications of each external service, including communication end-point, port type, operation, and message formats
- Communication style for each interaction with an external entity
- Optional communication order with external entities

The aggregation tool should have a simple command-line interface. It will analyze the WSDL description of each service, extract necessary binding information, combine it with the entity type, choose interaction style and generate a new aggregation process along with its WSDL definition.

4.3. Concept of Context Aggregation Processes

The goal of this section is to introduce an extension to Integration Process Architecture Pattern to support information aggregation in service compositions. A possible solution would be modeling aggregation as a set of processes integrated as a separate layer into the Integration Process Architecture Pattern.

Figure 4.2 demonstrates the general idea of the extension to Integration Process Architecture Pattern to support information aggregation. Context aggregation processes are basically located between domain independent and Domain Specific CIPs, but may also include CIPs from both layers. The following reasons underlie this particular placement:

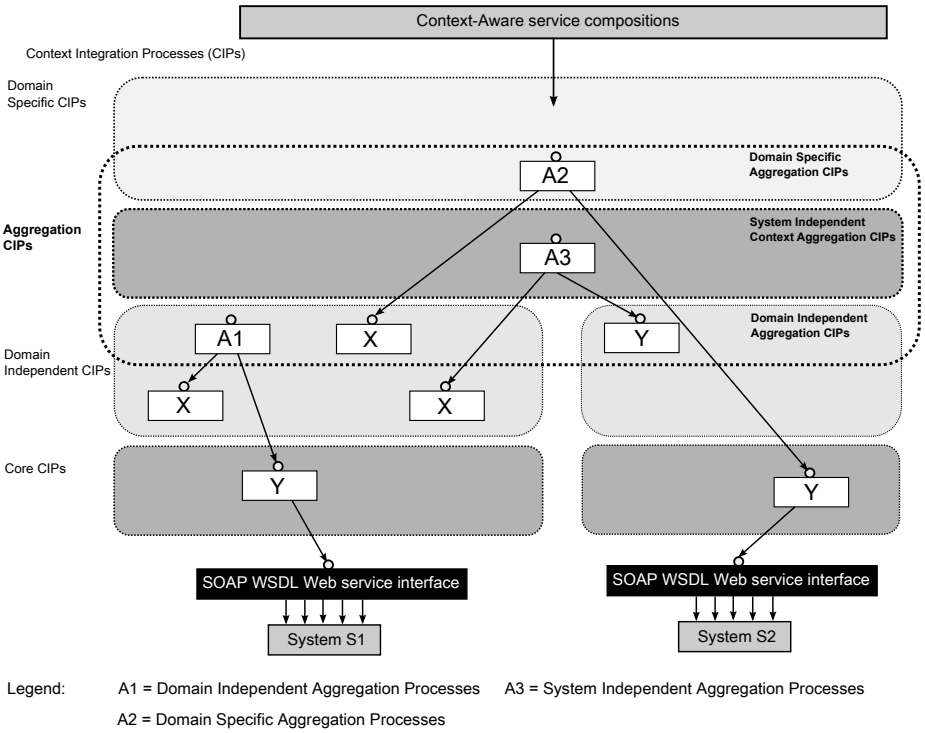

Figure 4.2.: Aggregation – Extension to Integration Process Pattern

- *Domain Independent Aggregation CIPs* In general, the context provisioning systems provide access to context management and other value added services through multiple interfaces which are encapsulated into the CIPs according to the Integration Process Pattern. It is thinkable that CIPs at the domain independent level may combine other CIPs to achieve finer grained selection of context information or supply new value added functionality. Hence, these new Domain Independent CIPs act as context aggregation processes.
- *Domain Specific Aggregation CIPs* Context-aware applications may integrate various context provisioning systems in order to obtain access to their context management services. In case the context information is required by a particular application implies the processing of heterogenous context data acquired from various sources, the corresponding processing logic can be encapsulated, moved out of the business logic, and represented as Domain Specific CIPs. These new Domain Specific CIPs can be expressed as context aggregation processes.

- *System Independent Context Aggregation CIPs* In course of time, Domain Specific CIPs aggregating information from multiple context provisioning systems may evolve and be shared between different context-aware applications. Thereby, these context aggregation processes are moved out of the level of Domain Specific CIPs and placed at their appropriate level between Domain Independent CIPs of various context provisioning systems and Domain Specific CIPs of different context-aware applications.

The naming of the context aggregation processes is affected by the current placement of a particular process and specific aggregation logic expressed in a process. This way, context aggregation processes should either have a name describing their functionality or use aggregated names of the CIPs they are using.

In the following, Google Maps Web services are integrated using Context Integration Processes to demonstrate how the proposed concept can be applied to aggregate services provided by C-CAST CMF and Google Maps Web services.

4.4. Google Maps Web Services

This section discusses Google Maps Web services, in particular the Google Geocoding and Google Directions services. The first part of this section is devoted to the brief specification of the RESTful interfaces to access the Google Maps services over HTTP. Subsequently, their integration into service compositions is realized using Context Integration Processes.

4.4.1. Usage Limits

According to the usage limits [56] introduced by the Google Maps Web Service API, Google services may only be used in conjunction with a Google map. Hence, Google Geocoding and Directions results should be displayed on a Google map. Other use is prohibited. In order to use and display the content provided by Google Maps services without a corresponding Google map, a written permission from Google is required.

Furthermore, the Google Maps API Premier sales team [57] should be contacted to obtain a commercial license, if taxi booking service based on the taxi service provider intends to charge users any fees for the usage.

4.4.2. Google Geocoding Web service

Google Geocoding Web service supplies the functionality to convert the postal addresses into geographic coordinates expressed as latitude / longitude value pair. Additionally, Google Geocoding supports converting geographic coordinates back to their representation as postal addresses. This service is known as *Reverse Geocoding*. Both Google Geocoding services are accessible through RESTful Web service interface described in the following paragraph.

Google Geocoding RESTful Web Interface A Google Geocoding request should be of the following form:

```
http://maps.googleapis.com/maps/api/geocode/OUTPUT?PARAMETERS
```

Where `OUTPUT` may be either of the following values:

- `json` returns information represented in JavaScript Object Notation (JSON) format
- `xml` returns information represented in XML format

The arguments passed to the Google Geocoding service are represented as `PARAMETERS` separated with the ampersand `&` character. Following `PARAMETERS` are expected:

- `address` is a required parameter representing the address information to geocode.
- `latlng` is a required parameter containing comma separated latitude and longitude values to geocode.
- `bounds` is an optional parameter representing the bounding box of the viewport within which to bias geocode results more prominently.
- `region` is an optional parameter representing the region code expressed as a top level domain (ccTLD) two-character value.
- `language` is an optional parameter indicating the language in which the result should be returned.
- `sensor` is a required parameter specifying if the request comes from a device with a location sensor. The value of this parameter must be `true` or `false`.

To invoke the Google Geocoding service, the postal address details should be passed in `address` parameter. Alternatively, to access Google Reverse Geocoding the geographic coordinates are passed in `latlng` parameter.

4.4.3. Google Directions Web service

Google Directions Web service provides functionality to calculate route directions, distance and transportation duration between different locations expressed in terms of postal addresses or geographical coordinates. The Google Directions service support various transportation modes including *driving*, *walking*, or *bicycling* modes. Furthermore, the route directions can be calculated through specified locations, so-called *waypoints*, or by specifying route restrictions such as avoiding highways and toll roads. Finally, Google Directions service may return multiple alternative routes between different locations if needed. Google Directions service is accessible through a RESTful Web service interface described in the following paragraph.

Google Directions RESTful Web Interface A Google Directions request should be of the following form:

```
http://maps.googleapis.com/maps/api/directions/OUTPUT?PARAMETERS
```

Where `OUTPUT` may be either of the following values:

- `json` returns information represented in JavaScript Object Notation (JSON) format
- `xml` returns information represented in XML format

The arguments passed to the Google Directions service are represented as `PARAMETERS` separated with the ampersand `&` character. Following `PARAMETERS` are expected:

- `origin` is a required parameter containing the origin address.
- `destination` is a required parameter containing the destination address.
- `mode` is an optional parameter specifying the transport mode used to calculate directions. Available modes are *driving*, *walking*, or *bicycling*. The default value is *driving*.
- `waypoints` is an optional parameter used to specify special locations on the route. Each waypoint is represented as latitude / longitude coordinate or as an address.
- `alternatives` is an optional parameter indicating if the Google Directions service should provide more than one route alternative in the response.
- `avoid` is an optional parameter specifying if the indicated features should be avoided during the route calculation. Current features include *tolls* (toll roads and bridges) and *highways*.
- `units` is an optional parameter specifying the unit system used to express results. Following values can be passed: *metric* for metric system and *imperial* for the English system.

- `region` is an optional parameter representing the region code expressed as a top level domain (ccTLD) two-character value.
- `language` is an optional parameter indicating the language in which the result should be returned.
- `sensor` is a required parameter specifying if the request comes from a device with a location sensor. The value of this parameter must be `true` or `false`.

Address information passed in the `origin` and `destination` parameters might be expressed as a comma separated list of latitude and longitude values or as a postal address. In the former case, no space is allowed between the latitude and longitude values.

4.4.4. Realization of Google CIPs

This section describes the realization of the Context Integration Processes for Google Geocoding and Google Directions services.

Context Provisioning Layer

In a similar manner as C-CAST CMF, the RESTful interfaces of Google Geocoding and Directions Web services were wrapped into standard Web service interface exposed through WSDL and accessed via SOAP over HTTP. Apache CXF [49] was taken for the realization of the SOAP Web service interface as the best suited JAX-WS [48] and JAX-RS [51] reference implementation.

According to the overview provided in the previous section, the following list of services was wrapped into SOAP Web service interfaces (see Figure 4.3):

- `GoogleDirections` service
- `GoogleGeocoding` service

Top down development was chosen for the realization of the Web service proxy server for Google Maps Web services. The first step was to specify available interfaces of Google Geocoding and Directions Web services using standard WSDL. Additionally, an XML Schema data model was defined to represent exchanged message formats and corresponding data types. Separate WSDL interface specifications were developed for Google Geocoding and Google Directions services. The next step was to generate the server side code using the *WSDL to Java* tool provided by Eclipse [52]. Finally, each method representing a separate Google service was implemented as a HTTP client to communicate with the corresponding RESTful interface. It is worthy noting that the XML formatted response messages generated by Google Maps Web services do not contain any XML namespaces. Hence, XSLT stylesheets were additionally defined to transform incoming replies by adding XML namespace attributes to contained elements.

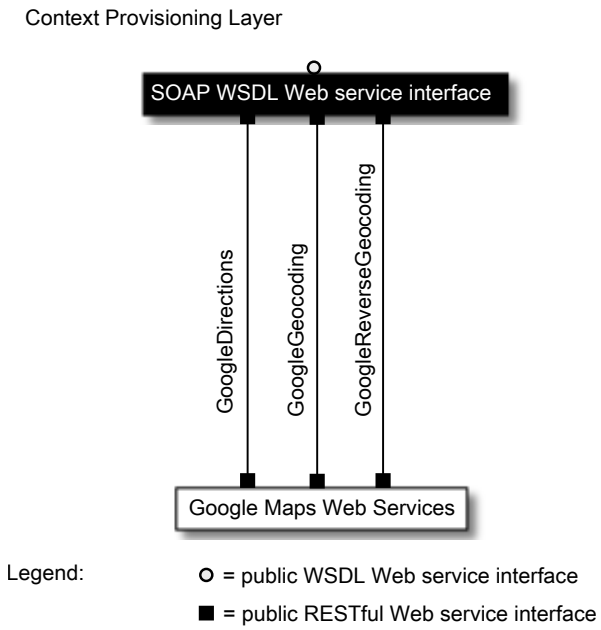

Figure 4.3.: Google Maps Web Services – SOAP WSDL Web Service Interface

To facilitate fault handling in the upper layers of the Context Integration Processes each Google Maps SOAP Web service returns a fault message in case of an invocation failure or other errors. Generated fault messages contain error code, response status, and further elements describing the actual cause of the invocation failure. As described in Section 3.3.1 on page 44, while generating SOAP Fault messages the CXF framework sets the default value for SOAP `faultcode` to `soap:Server`. In order to modify the SOAP Fault message generated by the CXF framework and store the real name of the thrown fault in `faultcode`, a new *outbound fault interceptor* was implemented as the extension to CXF *AbstractSoapInterceptor* and subsequently added to the outbound fault chain.

Core CIPs

Each SOAP Web service interface of Google Geocoding and Directions services was encapsulated into one concrete context integration process. Context integration processes were named according to the Google services they represent. The integration processes at this level encapsulate all services currently provided by Google Maps Web services and are illustrated in Figure 4.4.

- *GetGoogleDirections* integrates the querying functionality of the Google Directions service. To obtain information an origin and destination addresses along with other optional parameters should be provided. The *GoogleDirections* message format is used for the query result presentation.

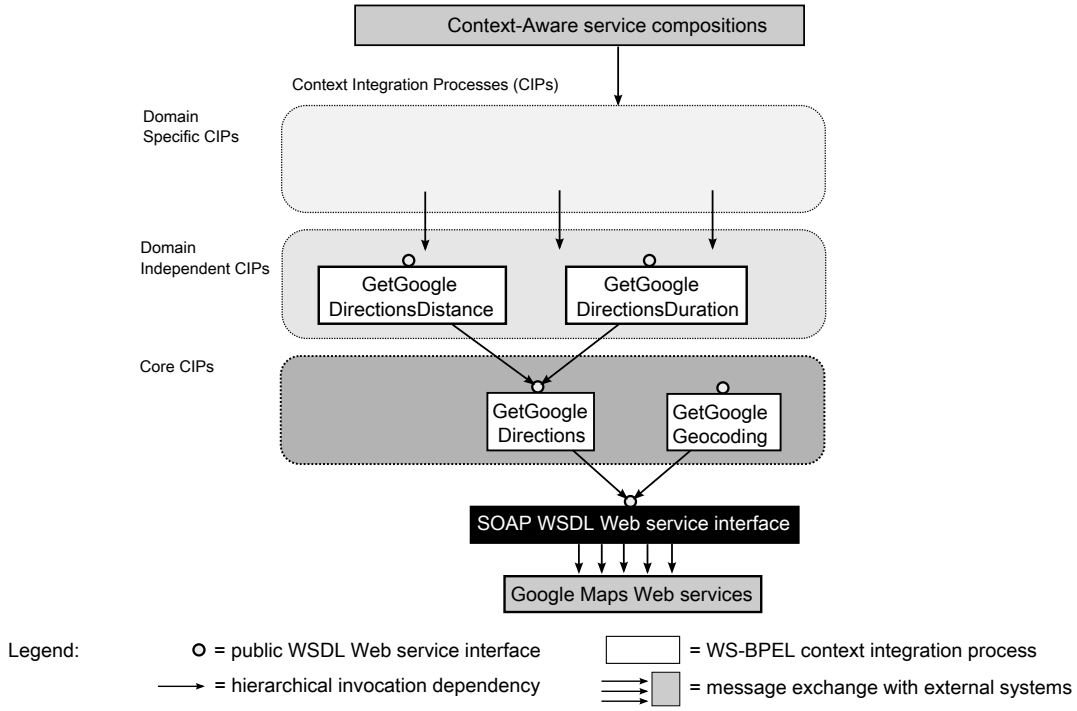

Figure 4.4.: Google Maps Web Services – Context Integration Processes

- *GetGoogleGeocoding* integrates the querying functionality of the Google Geocoding service interface. This integration process is used to geocode postal addresses as well as to transform geographic coordinates into postal address representation. GetGoogleGeocoding integrates the *GoogleGeocoding* message format for result presentation.

The integration processes at this level encapsulate all interfaces currently exposed by Google Geocoding, Reverse Geocoding, and Directions Web services.

Domain Independent CIPs

Domain Independent CIPs refine the information returned by the Core CIPs integrating Google Maps Web services. As Domain Independent CIPs the following integration processes were defined and implemented (see Figure 4.4).

- *GetGoogleDirectionsDuration* is a specialization of the Core CIP *GetGoogleDirections* used to obtain the route duration. Since response message may contain many alternative routes, this CIP calculates the shortest duration for the route and blanks out all other returned elements.

- *GetGoogleDirectionsDistance* is another specialization of the Core CIP *GetGoogleDirections* used to obtain the route distance. Since response message may contain many alternative routes, this CIP calculates the shortest distance for the route and blanks out all other returned elements.

4.5. Realization of Context Aggregation Processes

4.5.1. Domain Independent Context Aggregation Processes

Since most of the context aggregation processes are tailored to the specific requirements of the particular application domain, this section defines only Domain Independent aggregation processes which can be shared among various domains.

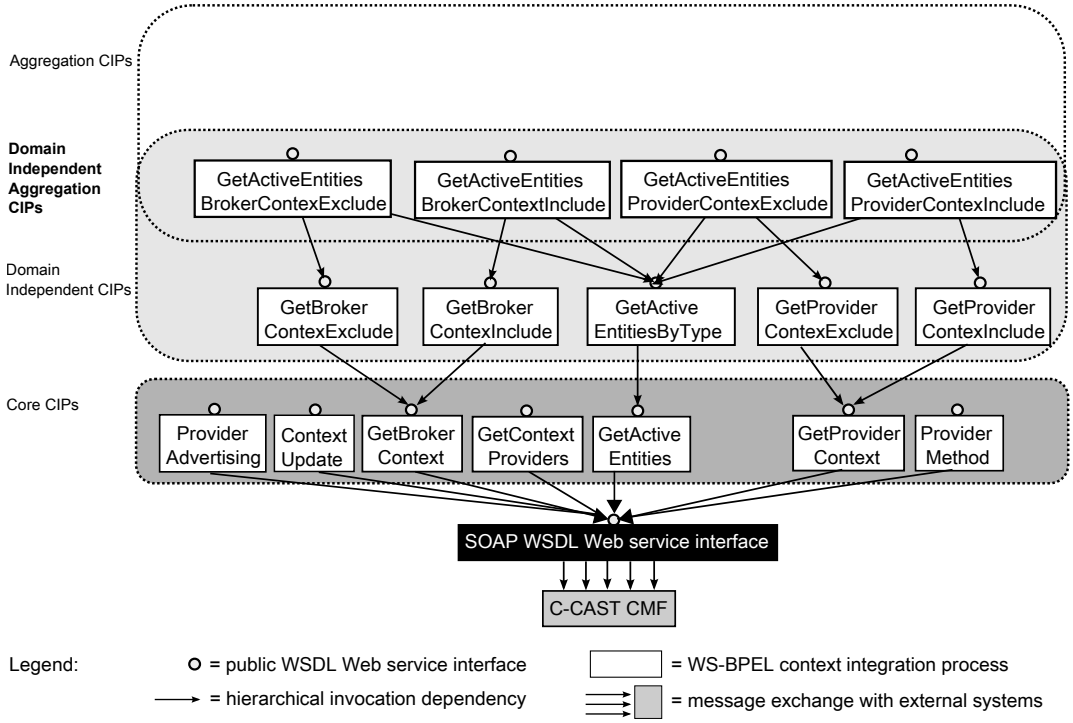

Figure 4.5.: C-CAST CMF – Domain Independent Context Aggregation Processes

As shown in Figure 4.5, the following Domain Independent aggregation processes were defined and implemented at this layer:

- *GetActiveEntitiesBrokerContextInclude* represents a context aggregation process which relies on the Domain Independent CIPs *GetActiveEntitiesByType* and *GetBrokerContextInclude*. This aggregation process is used to retrieve specific context information for a set of active entities which belong to a particular *Entity* type. *GetActiveEntitiesBrokerContextInclude* specifies a restriction on parameters that hold the *Entity* type, *Context Scope* names and other context attributes which should be contained in the result set.
- *GetActiveEntitiesBrokerContextExclude* represents a context aggregation process which relies on the Domain Independent CIPs *GetActiveEntitiesByType* and *GetBrokerContextExclude*. This aggregation process is used to retrieve a specific context information for a set of active entities which belong to a particular *Entity* type. *GetActiveEntitiesBrokerContextExclude* specifies a restriction on parameters that hold *Entity* type, *Context Scope* names and other context attributes which should not be contained in the result set.
- *GetActiveEntitiesProviderContextInclude* represents a context aggregation process which relies on the Domain Independent CIPs *GetActiveEntitiesByType* and *GetProviderContextInclude*. This aggregation process is realized in the same way as the related context aggregation process of the Context Broker. It is used to retrieve a specific context information provided by a particular Context Provider.
- *GetActiveEntitiesProviderContextExclude* represents a context aggregation process which relies on the Domain Independent CIPs *GetActiveEntitiesByType* and *GetProviderContextExclude*. This aggregation process is realized in the same way as the related context aggregation process of the Context Broker. It is used to retrieve a specific context information provided by a particular Context Provider.

4.5.2. System Independent Context Aggregation Processes

According to the aggregation concept outlined in the previous chapter, context aggregation processes can be defined at three different layers: *Domain Independent Aggregation CIPs*, *Domain Specific Aggregation CIPs* and at the intermediate *System Independent Context Aggregation Processes* layer.

Located at the intermediate layer, System Independent context aggregation processes integrate context data retrieved from various context provisioning systems and are used to disseminate transformed knowledge to different context-aware applications. As shown in Figure 4.6, the following aggregation processes were defined and implemented at this layer:

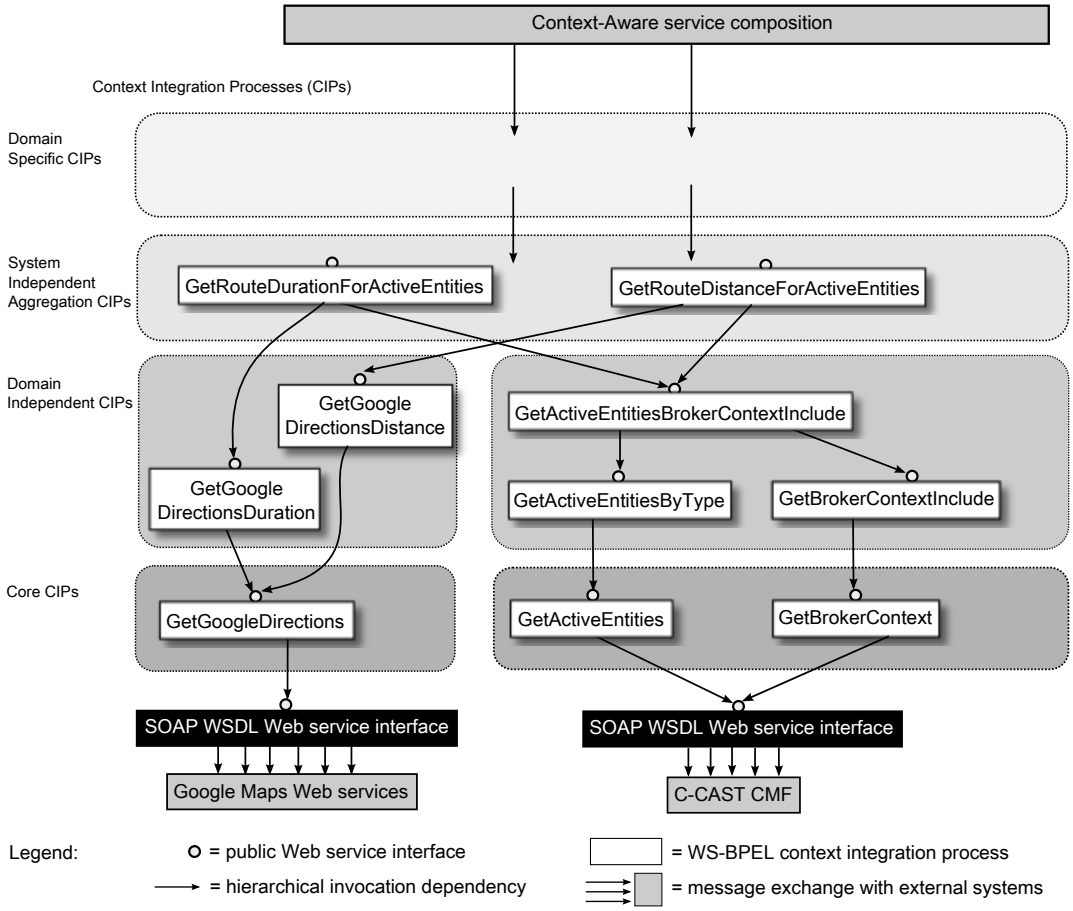

Figure 4.6.: System Independent Context Aggregation Processes for C-CAST CMF and Google Maps Web Services

- *GetRouteDurationforActiveEntities* calculates time required for each entity to travel the distance between its current location and a specified destination address. This aggregation process can be shared among different application domains and relies on Domain Independent CIPs *GetGoogleDirectionsDuration* and *GetActiveEntitiesBrokerContextInclude* processes. *GetRouteDurationforActiveEntities* expects the destination address and entity type as input parameters and returns a list of available entities combined with the related route duration.
- *GetRouteDistanceforActiveEntities* calculates the distance between the active entity's current location and a specified destination address. This aggregation process can be shared among different application domains and relies on Domain Independent CIPs *GetGoogleDirectionsDistance* and *GetActiveEntitiesBrokerContextInclude* processes. *GetRouteDistanceforActiveEntities* expects the destination address and entity type as input parameters and returns a list of available entities combined with the related route distance.

Chapter 5

Evaluation

The concepts describing integration (see Chapter 3) and aggregation (see Chapter 4) of context information into service compositions were presented and discussed in the previous chapters of this thesis. This chapter is devoted to the evaluation of the proposed solution. At the beginning of this chapter, Domain Specific context integration are developed to complete the integration of the C-CAST context management framework and Google Maps Web services into the sample service composition.

The developed context integration processes are meant to be a proof of the concepts presented in the previous chapters. After a short introduction of the available service composition engines, context integration processes will be deployed to Apache ODE [58] and OW2 Orchestra [59] composition engines to present interoperability and engine implementation independency.

5.1. Domains Specific Integration Processes

Taxi Service Provider, the sample use case scenario presented in Section 1.1 on page 2, is used to evaluate the context integration and aggregation concepts. In order to complete the integration of C-CAST CMF and Google Maps Web services into the sample Taxi Service Provider process the Domain Specific CIPs should be defined. Figure 5.1 demonstrates the proposed integration processes which are subsequently described in detail in the corresponding parts of this section.

Domain Specific CIPs are used to integrate Domain Independent and aggregated functionality provided by the context provisioning systems and other systems into service compositions. The following Domain Specific CIPs were defined to complete the implementation of the Taxi Service Provider process:

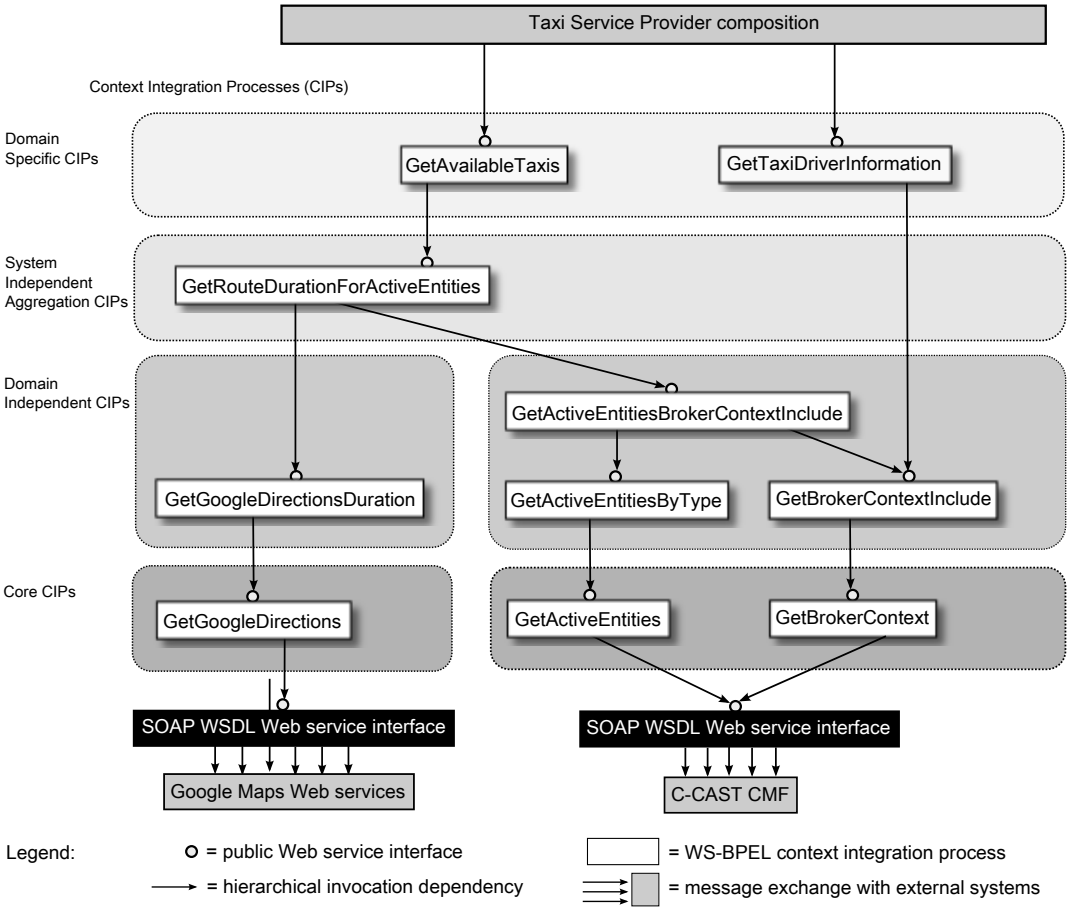

Figure 5.1.: Taxi Service Provider Context Integration Processes

- *GetAvailableTaxis* queries available taxi cabs, obtains their current location, and calculates time required for each taxicab to reach customer destination. This Domain Specific process returns a list containing up to 5 entries representing available taxicabs next to the customer. This CIP gets an input parameter that holds the current customer location used to find the nearest taxis. *GetAvailableTaxis* relies on the System Independent Context aggregation CIP *GetRouteDurationforActiveEntities*.
- *GetTaxiDriverInformation* returns the profile information of a particular taxi driver including contact details. This context integration process accepts an input parameter that holds the information identifying a particular taxi driver. *GetTaxiDriverInformation* relies on the Domain Independent CIP *GetBrokerContextInclude* of the C-CAST CMF.

5.2. Composition Engines

Since the proposed context integration approach does not involve the modification of the WS-BPEL standard, the following open-source service composition engines can be used for the evaluation of interoperability and engine implementation independency.

5.2.1. Apache Orchestration Director Engine

Apache Orchestration Director Engine (ODE) [58] is an open source workflow engine implemented as a compact and embeddable software component capable of managing the execution of both short and long-running business processes defined using the BPEL process description language. It is a community driven project under the supervision of the Apache Software Foundation.

Apache ODE provides support for both WS-BPEL 2.0 OASIS standard and the legacy BPEL4WS 1.1 vendor specification. Additionaly, Apache ODE extends the WS-BPEL in areas not covered by the specification, e.g *External Variables* extension which appears to be important with the reference to current thesis statement. External variables extension enables sharing data between the process and external systems, by treating those independent entities as BPEL variables that can be used in expressions and assignments.

Apache ODE supports different communication layers such as the HTTP transport communication layer based on Axis2 and another one based on the Java Business Integration (JBI) standard. Finally, Apache Ode has a complete management API to check which processes are deployed, running and completed instances, variables values and more.

5.2.2. OW2 Orchestra

Founded in January 2007 as a result of the merger of ObjectWeb and Orientware communities, the OW2 consortium is an international open source community which includes R&D teams, industrials partnerships and research organization. OW2 Orchestra [59] is an open source business process management (BPM) platform providing orchestration functionalities to handle long-running, service oriented business processes. Available under the LGPL license Orchestra is a multi platform and multi database workflow engine providing support for WS-BPEL 2.0 OASIS standard.

Build on top of the generic *Process Virtual Machine* OW2 Orchestra represents an embeddable, pluggable and extensible software component deployable on different servlet containers (Tomcat, Swing, etc) and Java EE servers (JOnAS, JBoss, Weblogic, etc). Using Apache Camel and Petals ESB , the open source ESB frameworks, as transport for Web services interactions OW2 Orchestra handles service interaction in a heterogeneous environment supporting various communication layers such HTTP, FTP, Mail, JMS, File, Presto, etc.

Finally, OW2 Orchestra offers graphical BPM console based on Web 2.0 and Ajax technologies for design, deployment, management and monitoring of business processes. The graphical environment of OW2 Orchestra provides modeling support of business processes using BPMN notation.

5.3. Evaluation

Integration of the context provisioning system C-CAST CMF was modeled as executable workflows realized in standard WS-BPEL. In order to ensure interoperability and engine implementation independency of the selected approach the developed WS-BPEL processes were deployed and evaluated using two different engines: Apache ODE and OW2 Orchestra.

To determine if the developed integration processes meet the requirements a set of test suites was defined using *soapUI* testing tool [60]. *soapUI* introduces web-testing capabilities that allow recording and replaying functional tests automatically. The developed tests were organized into *soapUI* projects containing multiple suites of functional tests to evaluate context integration processes of each particular integration layer. Each test suite addressed a particular context integration process, defined test requests for different service endpoints, evaluated standard and fault functionality using pre-defined assertions such as XML Schema compliance, not SOAP fault and SOAP response. While deployment of the context integration processes was in general successful, the following issues were discovered:

1. **Different Service Endpoint Address Declaration** Apache ODE and OW2 Orchestra do not support automatic generation of WSDL port type bindings and service endpoint declarations. Furthermore, both engines use different naming schemes for the service endpoint declaration.
   - *Apache ODE* expects `http://hostname:portname/ode/processes/ServiceName`
   - *OW2 Orchestra* expects `http://hostname:portname/orchestra/PortName`

2. **Different Variable Initialization** In order to access a variable's content in Apache ODE each variable should be pre-initialized using an in-line `from`-spec of an `assign` activity. Furthermore, to write values to a particular field of a variable this field must be previously initialized. In case of a complex variable structure, this initialization approach is confusing and leads to additional hidden errors. In contrast to this approach, the OW2 Orchestra engine supports automatic variable initialization. Hence, in OW2 Orchestra any variable field can be easily accessed and updated using a query language, such as XPath, without previous variable initialization.

3. **String value limitations in Orchestra** Expression values of type `xs:string` [55] are limited to 255 characters. Processing of String values exceeding this size causes freezing of process instance execution due to an internal database exception.

4. **No support in Orchestra for some XML element attributes** OW2 Orchestra does not provide support for the attribute `xml:space` [55] used in `bpel:literal` constructs. `xml:space` attribute may be attached to an element to signal an intention that white space used in that element should be preserved by applications. The usage of `xml:space` used in WS-BPEL `bpel:literal` constructs is shown in Listing 5.1.

Listing 5.1 Usage of the Attribute `xml:space` in WS-BPEL

```
<bpel:assign validate="no" name="Assign">
  <bpel:copy>
    <bpel:from>
      <bpel:literal xml:space="preserve">
        First Line
        Second Line
        Third Line
      </bpel:literal>
    </bpel:from>
    <bpel:to part="payload" variable="input">
      <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
        <![CDATA[tns:input]]>
      </bpel:query>
    </bpel:to>
  </bpel:copy>
</bpel:assign>
```

5. **Specific XSL transformation in Orchestra** In order to enable XSL transformation of variable's content attention must be paid to the namespace prefixes used in the XSLT sheets. XSLT sheets should use the same namespace prefixes as in the XML document passed to the XSLT sheet. Otherwise, an exception is thrown pointing out that a namespace prefix is not bound.

6. **Integer representation in Apache ODE** Apache ODE converts integer numbers to their rational representation. Hence, in order to use integer numbers in `foreach` loops they must be explicitly transformed using XPath function `number()`.

The identified issues were reported as bugs to the workflow engines' development teams. Since there was no feedback during the time period of this thesis and no fixes were provided in the new engine versions, the forced solution was to develop different versions of Context Integration Processes for each workflow engine. The deployed Context Integration Processes differ only in the way the outlined issues above were resolved. As for the rest, they are based on the WS-BPEL standard and do not leverage any extension mechanisms.

The implemented context integration solution has been evaluated to determine the degree to which the proposed concept fulfills the stated goals. According to the expectations, the numerous abstraction layers of the Integration Process Pattern along with the additional Aggregation Layer had a negative impact on the overall system performance. The invocation of the Domain Specific Context Integration Process *GetAvailableTaxis* takes currently about 5 seconds to complete and calculate result values both for Apache ODE and OW2 Orchestra engine. In all other respects, the concept of Context Integration Processes has approved itself to be the most suitable context integration strategy realizing the main objectives outlined in Section 1.2.

Enabling integration of context in WS-BPEL compositions
The Context Integration Processes approach enables integration of context information into service compositions realized in the standard WS-BPEL language.

Providing a composition engine independent solution
The Context Integration Processes approach relies on the existing web-services standards and frameworks and does not implicate the modification of WS-BPEL engine. Hence, integration processes can be deployed on various workflow engines which support standard WS-BPEL 2.0 specification.

Enabling encapsulation and outsourcing of context information aggregation
Processing and aggregation of context information is encapsulated, moved out of application logic and handled as a separate layer in the proposed integration architecture.

Providing integration support of different context provisioning systems
Context Integration Processes approach can be applied to integrate various context provisioning systems, even to those which do not expose SOAP WSDL interface.

Enabling loosely coupling to context integration system
Since Context Integration Processes approach defines the necessary system abstraction, the underlying context provisioning systems may be easily exchanged. Interfaces of the integrated context provisioning system may evolve over time without forcing context-aware applications to adapt their business logic to the introduced changes.

Providing component reusability
The Context Integration Processes approach separates context integration into hierarchy of different semantical CIPs which allows a high maintainability and reusability of the integration components.

Enabling integration system extensibility
Finally, a loosely coupled architecture enables integration of context provisioning systems to be more manageable. New context integration processes of different semantic levels can be built upon existing components and be exposed to context-aware applications.

Chapter 6

Context Modeling Tool

This chapter describes an extension of WS-BPEL modeling tools to support the development of context-aware compositions using the integration concept introduced in the previous chapters. Due to the variety of existing BPEL modeling tools, this chapter mainly focuses on the general specification details. It identifies application domains of the extension of a BPEL modeling tool, specifies user groups, their expectations and basic requirements, and describes use cases to outline the extension functionality. The specification details act as the base for design decisions and subsequent implementation. According to the current thesis statement, this chapter provides only the specification and design details of an extension of Eclipse BPEL Designer [61]. Hence, graphical support is not realized within the scope of this thesis.

6.1. Application Domains

This section identifies the two application domains of the context extension of a BPEL modeling tool. In general, the modeling extension should support modelers in the following domains:

- **Development of context-aware service compositions** The extension of a BPEL modeling tool should support modelers while using existing Domain Specific context integration processes in their context-aware service compositions.
- **Development of context integration processes** The extension of a BPEL modeling tool should support modelers while creating new Core, Domain Independent, and System Independent Aggregation CIPs to integrate or aggregate functionality provided by various context provisioning systems.

The remainder of this chapter describes each application domain in more detail and illustrates the required designer views along with the context modeling elements. Furthermore, it specifies user groups and defines use cases illustrating how the proposed modeling extension may support modelers while performing the respective activity.

6.2. Context Integration Processes Repository

In order to make context integration processes available for service compositions, a Context Integration Processes repository (CIPs repository) is required. Modelers access the CIPs repository to share context integration processes and use them in service compositions. Context integration processes available in the CIPs repository should be deployed and ready for use.

6.3. Context-Aware Service Composition Modeling

6.3.1. Actors

The starting point is to identify actors exploiting the proposed context extension of a BPEL modeling tool in this application domain and describe their expectations and requirements.

The primary actors in this application domain are responsible for the modeling of service compositions, integrating context information to solve their business needs. Hence, modelers access the CIPs repository to obtain a list of Domain Specific context integration processes and use them in service compositions.

The context extension of a BPEL modeling tool should simplify the development of context-aware applications by providing a graphical support and BPEL code generation. Users of this group expect to work using traditional designer views of the integrated development environment dedicated to business process modeling. Hence, a proposed extension of a BPEL modeling tool should rely on the existing design capabilities, add context integration support, and introduce no fundamental modifications to the traditional BPEL designer view. For the sake of convenience, modelers of context-aware service compositions are referred as ***composition modelers*** in the rest of this section.

6.3.2. User Interfaces

User interfaces define the interaction space between composition modelers and the context extension. According to the previously introduced requirements, composition modelers expect to work using the traditional modeling view of a BPEL modeling tool.

Figure 6.1 illustrates a common BPEL modeling view with the additional palette containing graphical elements to support context-awareness. The following elements comprise this graphical perspective:

1. **Graphical Editor** is an editor area containing the primary BPEL process content represented as graphical elements.

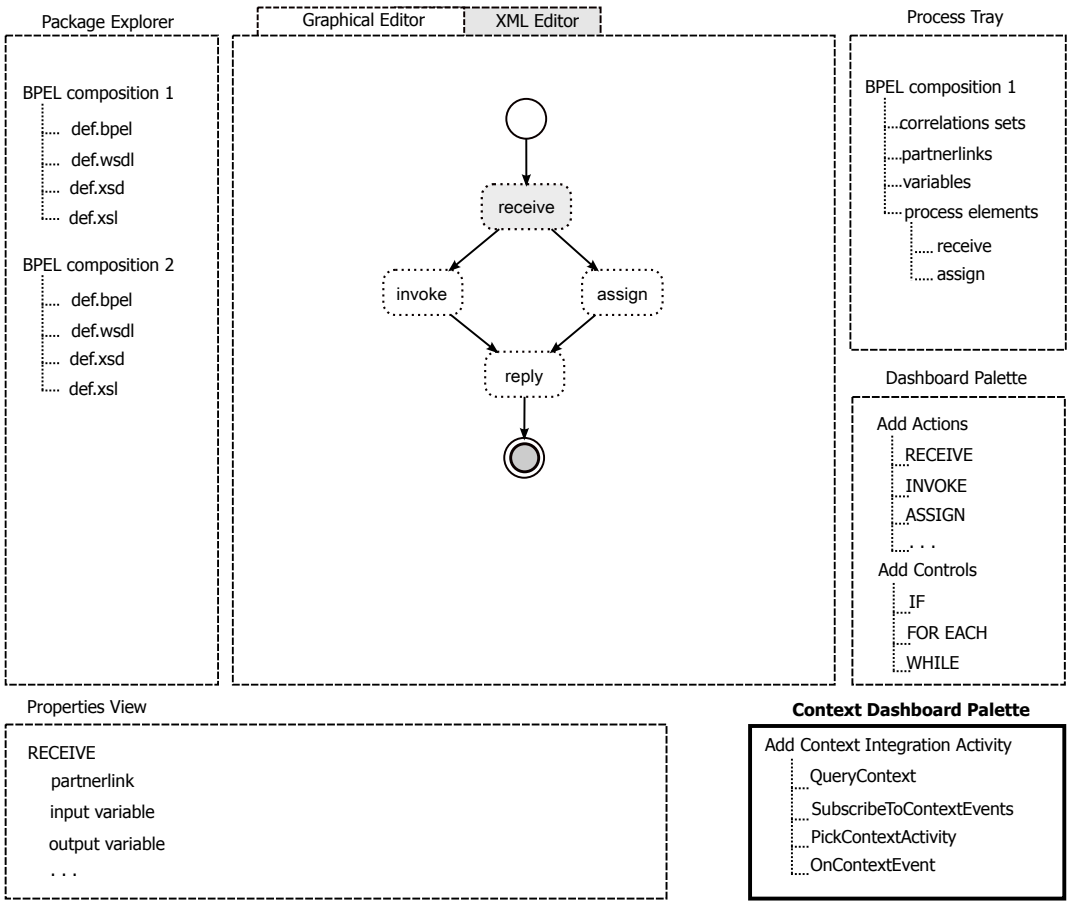

Figure 6.1.: WS-BPEL Modeling Tool Extension – Context Dashboard Palette

2. **XML Editor** is an editor area containing the primary BPEL process content described in XML.
3. **Package Explorer** reveals current BPEL package contents such as the BPEL process file, WSDL interface description files, XSD and XSLT documents, and other artifacts required for BPEL process deployment.
4. **Dashboard Palette** is an area listing BPEL activity elements available for the current BPEL modeling tool.
5. **Process Tray** contains the elements contained in the current BPEL process such as partner links, correlation sets, variables, etc.
6. **Properties View** displays properties and other details of the currently selected BPEL process element.

7. **Context Dashboard Palette** is an additional area listing context-aware elements used to integrate context information into the BPEL process.

The main objective is to extend a BPEL modeling tool with the additional Context Dashboard Palette.

6.3.3. Use Cases

The basic use case illustrating context integration into service compositions is described in the following.

Development of Context-aware Service Compositions This use case is abstract and defines the development of context-aware service compositions. It describes the standard course of actions performed by a composition modeler and specifies the behavior of the modeling extension to support code generation while adding context activity elements to the designer view.

Preconditions: The repository of context integration processes is bound and contains at least one Domain Specific context integration process. The basic BPEL structure of a business process is defined and contains no errors. A process variable used to store retrieved context information is defined and optionally initialized. A graphical perspective extended to support development of context-aware service compositions is opened.

Postconditions: The BPEL description of a business process contains the context-awareness logic, is validated, and contains no errors. The necessary WSDL and XSD files of corresponding context integration process are copied into the business process directory, are validated, and contain no errors.

Basic course of actions:

1. The composition designer selects a *new context activity* element from the Context Dashboard Palette and places it in Graphical Editor area.
2. The extension of a BPEL modeling tool displays the graphical representation of a context activity element in the Graphical Editor area and generates the BPEL code for the corresponding standard BPEL elements.
3. The composition designer selects the *context activity* element in the Graphical Editor area to show and edit its information in Properties View.
4. The composition designer selects a Domain Specific CIP from the list of available context integration processes.
5. The composition designer provides necessary details to complete the implementation of the context activity element.

6. The extension of a BPEL modeling tool creates copies of WSDL and XSD files of the Domain Specific CIP in the directory of current business process.
7. The modified BPEL process file is validated.

6.3.4. Design and Background Knowledge

This section provides the design details of the proposed modeling extension for the Eclipse BPEL Designer [61] to support development of context-aware service compositions using the concept of Context Integration Processes. The primary target user group of this extension includes composition modelers. At the beginning of this section, a brief introduction of the main technologies used for the implementation is done to give the reader the hands-on knowledge necessary for a better understanding. The next part covers the design details of the extension of the Eclipse BPEL Designer.

Background Knowledge

Eclipse Eclipse [52] is an open development platform comprised of extensible frameworks, tools, and runtimes for building and managing software across the lifecycle. The Eclipse platform is organized based on the concept of *plug-ins* - software modules contributing to the system functionality. The modular architecture of Eclipse provides integrated resource management and simplifies the installation of additional features and plug-ins into an Eclipse.

Eclipse Extension Points and Extensions The Eclipse platform defines a comprehensive extension model to enable contribution to the basic platform behavior. The functionality of existing plug-ins can be customized or even extended by other plug-ins through the mechanism of *extension points* and *extensions* [62]. Plug-ins which declare so-called *extension points* allow other plug-ins to contribute to their functionality by providing suitable *extensions*. To achieve this an extension point defines a type of contract that extensions must conform to. This contract must be implemented in the extensions provided by other plug-ins connecting to this extension point.

Eclipse Modeling Framework The Eclipse Modeling Framework (EMF) [63] represents a modeling framework and code generation facility for building tools and other applications based on a structured data model. Eclipse EMF provides means to define a data meta model using XML Metadata Interchange, Java annotations, UML, or XML Schema. Once a structured meta model is specified, Eclipse EMF generates a corresponding set of Java implementation classes. Additionally, Eclipse EMF supports model change notifications, provides persistence support, facilitates model validation, etc.

Graphical Editor Framework The Graphical Editing Framework (GEF) [64] provides technology to create graphical editors and views for the Eclipse Workbench UI. GEF offers an interactive model-view controller framework fostering the implementation of Eclipse graphical editors, a visualization toolkit which enables implementation of Eclipse graphical view, and a layout and rendering toolkit for displaying graphics on an SWT Canvas.

Eclipse BPEL Designer Realized as an Eclipse plug-in the BPEL Designer [61] is a contribution to the Eclipse platform to support the definition, authoring, editing, deploying, testing, and debugging of WS-BPEL 2.0 processes. The BPEL Designer project provides graphical support for BPEL process development lifecycle, enable validation and deployment of BPEL processes, and aims to support debugging during process execution. The EMF data model defined by the Eclipse BPEL Designer covers the standard WS-BPEL 2.0 Specification [2]. A GEF-based editor is used to provide a graphical means to design BPEL processes. Finally, the BPEL Designer leverages the standard Eclipse extension mechanisms to extend its functionality and support additional BPEL constructs and user defined features.

Context Integration Activities

To support composition modelers while integrating context information into service compositions, the Eclipse BPEL Designer model must be extended with the additional context integration activities. Context-aware service compositions obtain context information using different patterns of communication. Basically, the context information is retrieved as an invocation result of Domain Specific CIPs. Both synchronous and asynchronous invocation mechanisms can be used to query context information. Alternatively, context-aware service compositions support processing of context events either within or concurrently to the control flow of the business process. The following context integration activities were defined:

QueryContext This activity represents a synchronous invocation of a Domain Specific context integration process used to populate a process variable with the context information. Hence, the QueryContext activity is realized as a scope containing invocation logic along with the variable initialization logic as shown in Listing 6.1.

SubscribeToContextEvents This activity is a special case of QueryContext activity, since a subscribe operation represents a one way invocation of a Domain Specific context integration process. SubscribeToContextEvents is comprised out of a scope containing a one way invocation logic and input variable initialization logic as shown in Listing 6.2.

Listing 6.1 Implementation – QueryContext Activity

```
<bpel:scope name="QueryContextScope">
  <bpel:sequence name="QueryContext">

    <!-- Define scope variables used in communication with Domain Specific CIP -->
    <bpel:variable name="inputCIP" ... />
    <bpel:variable name="outputCIP" ... />

    <!-- Initialize CIP input variable -->
    <bpel:assign validate="no" name="PrepareInputToCIP">...</bpel:assign>

    <!-- Invoke Domain Specific CIP synchronously -->
    <bpel:invoke name="InvokeCIP">...</bpel:invoke>

    <!-- Process retrieved context information and populate BPEL process' context variable -->
    <bpel:assign validate="no" name="PopulateContextProcessVariable">...</bpel:assign>

  </bpel:sequence>
</bpel:scope>
```

Listing 6.2 Implementation – SubscribeToContextEvents Activity

```
<bpel:scope name="SubscribeToContextEventsScope">
  <bpel:sequence name="SubscribeToContextEvents">

    <!-- Define scope variable used in communication with Domain Specific CIP -->
    <bpel:variable name="inputCIP" ... />

    <!-- Initialize CIP input variable -->
    <bpel:assign validate="no" name="PrepareInputToCIP">...</bpel:assign>

    <!-- Invoke Domain Specific CIP asynchronously -->
    <bpel:invoke name="InvokeCIP">...</bpel:invoke>

  </bpel:sequence>
</bpel:scope>
```

PickContextEvent This activity is used to provide a callback interface for a Domain Specific context integration process to enable processing of context events within the control flow of the business process. Using basic WS-BPEL constructs the PickContextEvent activity (see Listing 6.3) is realized as a scope containing a pick activity to process context events within the control flow of the business process.

Listing 6.3 Implementation – PickContextEvent Activity

```
<bpel:scope name="PickContextEventScope">
  <bpel:sequence name="PickContextEvent">

    <bpel:pick name="PickContextEvent">

      <!-- Process context events from Domain Specific CIP -->
      <bpel:onMessage partnerlink="..." operation="...">

        <!-- Process retrieved context information and populate BPEL process' context
            variable -->
        <bpel:assign validate="no" name="PopulateContextProcessVariable">...</bpel:assign>

      </bpel:onMessage>
    </bpel:pick>

  </bpel:sequence>
</bpel:scope>
```

OnContextEvent This activity is used to provide a callback interface for a Domain Specific context integration process to enable processing of context events concurrently to the control flow of business process. As shown in Listing 6.4, OnContextEvent is an extension of the EventHandlers implemented as onEvent activity.

Listing 6.4 Implementation – OnContextEvent Activity

```
<bpel:eventHandlers>

  <!-- Implement OnContextEvent as onEvent activity -->
  <bpel:onEvent partnerLink="..." portType="...">
    <bpel:scope name="OnContextEventScope">

      <!-- Process retrieved context information and populate BPEL process' context variable
          -->
      <bpel:assign validate="no" name="PopulateContextProcessVariable">...</bpel:assign>
    </bpel:scope>
  </bpel:onEvent>
  ...

  </bpel:eventHandlers>
</bpel:process>
```

Design

The starting point is to define an EMF model representation of each context integration activity, plug the custom activities into the BPEL model, and take care for the serialization and de-serialization. The next step is to visualize the context integration activities in the Eclipse UI. Besides visualization of context integration activities in the main BPEL Editor canvas, this includes the creation of a special Dashboard palette to group the context integration activities. Additionally, the BPEL Properties View must be extended. In general, two separate Eclipse plug-ins will be created: the **BPEL Designer EMF Model** plug-in and the **BPEL Designer UI** plug-in.

BPEL Designer EMF Model Extension As previously mentioned, the BPEL Designer leverages Eclipse EMF to model BPEL activities. Hence, to define new context integration activities an extension of the BPEL EMF model is required. Figure 6.2 illustrates the developed EMF model for the context integration activities.

WS-BPEL 2.0 standard specifies a special extension activity to extend WS-BPEL with additional language constructs. Since the BPEL Designer fully relies on the WS-BPEL 2.0 standard, it exposes the only extension points which conform to the BPEL specification. Therefore, context integration activities have to be implemented as BPEL `<bpel:extensionActivity>` containing other standard BPEL constructs such as `<bpel:invoke>`, `<bpel:pick>`, and `<bpel:assign>`. This ensures that the deployment of context-aware compositions will be supported by the most standard conform BPEL composition engines.

The following brief description identifies the implementation steps of the EMF model extension of BPEL Designer according to the course of action described in [65]:

1. Creation of a new Eclipse plug-in project *org.eclipse.bpel.context.model* implementing the extension point *org.eclipse.emf.ecore.generated_package*.
2. Creation of a new EMF *context.ecore* model associated with the BPEL EMF *bpel.ecore* model provided by BPEL Designer.
3. Definition of the new context integration activities as BPEL extension activities.
4. Creation of a corresponding EMF *context.genmodel* and subsequent generation of Java classes implementing the data model.
5. Implementation of reconciling methods *adoptContent()* and *orphanContent()* to enable BPEL Designer Editor canvas to react and display the changes introduced to the BPEL model.

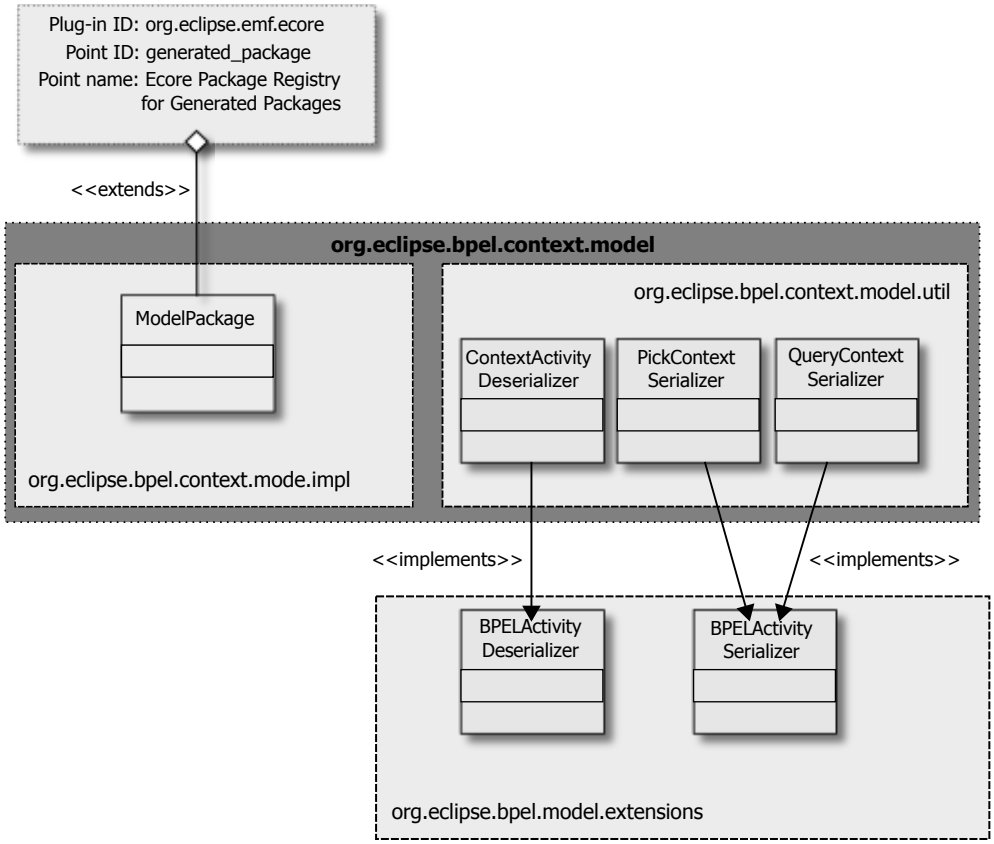

Figure 6.2.: Eclipse BPEL Designer Model Extension

6. Implementation of custom *Serializer* and *Deserializer* for each context integration activities to enable transformation of Java object representation to XML and vice versa. The Serializer is additionally used to create the overall structure of the context integration activities.

BPEL Designer UI Extension To separate user interface implementation from model representation an additional Eclipse plug-in project is required implementing the graphical extension points of BPEL Designer. The current BPEL Designer UI plug-in must implement the extension points exposed by the BPEL Designer to enable the representation of context integration activities on the Editor canvas and Dashboard Palette. Additionally, a special Properties View is required to display and edit properties for each context integration activity.

As shown in Figure 6.3, the BPEL Designer UI extension must implement the following extension points:

- *org.eclipse.bpel.ui.uiObjectFactories* extends the BPEL Designer to display the context integration activities on the BPEL Designer Editor canvas.
- *org.eclipse.bpel.common.ui.paletteAdditions* extends the BPEL Designer to display the context integration activities on the Dashboard palette.
- *org.eclipse.ui.views.properties.tabbed.propertySections* extends the Eclipse Workbench to display and edit custom activity properties.

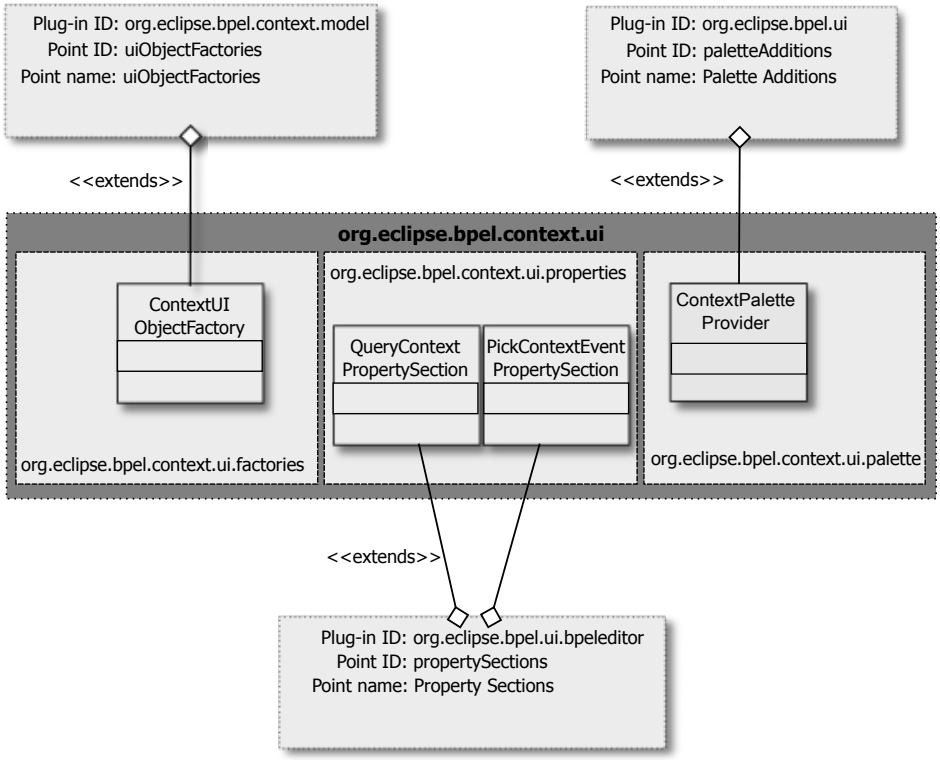

Figure 6.3.: Eclipse BPEL Designer UI Extension

The following brief description provides implementation details of the BPEL Designer UI extension:

1. Creation of a new Eclipse plug-in project: *org.eclipse.bpel.context.ui*.
2. Registering custom UIObjectFactory to create UI objects for context integration activities: *ContextUIObjectFactory*.
3. Creation of custom activity adapters for each context integration activity. Activity adapters describe how to access child activities within the custom activity and return a GEF representation of custom activities.

4. Registering custom IPaletteProvider to create a Dashboard palette for context integration activities: *ContextPaletteProvider*.
5. Registering new BPEL property sections in Eclipse Properties View for each context integration activity.

Context Integration Activities' Properties

Once a context integration activity is placed onto the BPEL Designer Editor canvas, a composition modeler is prompted to provide additional information required to complete the activity implementation. Using the BPEL Designer Properties View the composition modeler specifies the following information:

QueryContext activity

- Domain Specific context integration process name
- Domain Specific context integration process `partnerLinkType`, `partneRole`, `operation`
- Process variable used to store context information
- Initialize variable used as input to Domain Specific context integration process
- Initialize process variable using Domain Specific context integration process output variable

SubscribeToContextEvents

- Domain Specific context integration process name
- Domain Specific context integration process `partnerLinkType`, `partneRole`, `operation`
- Process variable used to store context information
- Initialize variable used as input to Domain Specific context integration process

PickContextEvent

- Domain Specific context integration process name
- Domain Specific context integration process output message type
- Process variable used to store context information
- Initialize process variable using Domain Specific context integration process output variable

OnContextEvent

- Domain Specific context integration process name
- Domain Specific context integration process output message type
- Process variable used to store context information
- Initialize process variable using Domain Specific context integration process output variable

6.4. Context Integration Processes Modeling

6.4.1. Actors

The primary actors in this application domain include modelers responsible for the design and management of Context Integration Processes. Basically, the integration process development may be accomplished using existing capabilities of currently available BPEL modeling tools. However, due to the fact that the proposed context integration solution represents a complex hierarchical Web service stack, a dedicated designer view is required to provide a simplified view on the integration components and their organization. For the sake of convenience, modelers of context integration processes are referred as ***integration modelers*** in the rest of this section.

6.4.2. User Interfaces

User interfaces define the interaction space between integration modelers and the context extension. To support integration modelers while integrating new context provisioning systems a new BPEL modeling perspective is required. Figure 6.4 demonstrates this modeling perspective containing editor areas, explorers, and element dashboard palettes required for simplified development of context integration processes:

1. **Graphical Editor** is an editor area comprised of the five layers according to the introduced context integration concept: *System Layer*, *Core Layer*, *Domain Independent Layer*, *System Independent Layer*, and *Domain Specific Layer*. Context integration and aggregation processes are organized, interconnected, and displayed according to their organization in the hierarchical integration stack.
2. **CIP Explorer** represents the CIPs repository and contains integration processes for each available context provisioning system. Context integration processes are organized in folders according to the layers they belong to.

3. **Integration Explorer** reveals the current context integration composition. Context integration processes involved into compositions are structured and organized to their placement in the hierarchical integration stack.
4. **CIP Dashboard Palette** is an area listing elements simplifying the integration of a new context provisioning system and the definition of new integration and aggregation processes.
5. **Properties View** displays properties, interfaces and other details of currently selected context integration process.

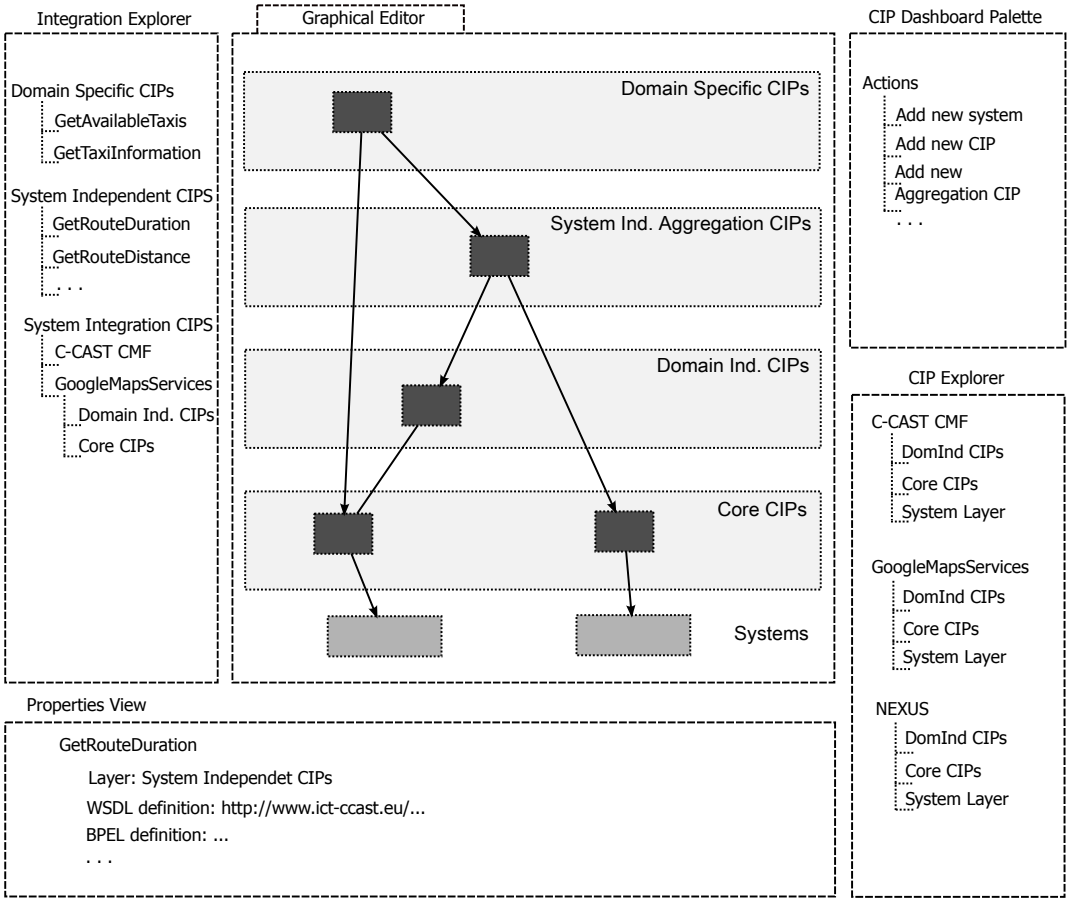

Figure 6.4.: WS-BPEL Modeling Tool Extension – CIP Modeling Perspective

This diploma thesis provides only specification details to this graphical perspective. An extension of a BPEL modeling tool to support integration of new context provisioning systems may be provided as future work in this area.

6.4.3. Use Cases

The following are the basic use cases describing integration of new context provisioning systems.

Integration of a Context Provisioning System This use case is abstract and defines the integration of a new context provisioning system. It describes the standard course of actions performed by an integration modeler and specifies the behavior of the modeling extension to support code generation.

Preconditions: The CIPs repository is bound and ready to store new context integration processes. The WSDL interface definition of the context provisioning system exists, is validated, and contains no errors. A graphical perspective extended to support development of context integration processes is opened.

Postconditions: The context integration processes for the new context provisioning system are defined and uploaded to CIPs repository.

Basic course of actions:

1. The integration modeler selects the graphical element *new context provisioning system* from the CIP Dashboard Palette and places it in Graphical Editor area.
2. The integration modeler selects a *context provisioning system* element in the Graphical Editor to show and edit its information in the Properties View.
3. The integration modeler provides the system's WSDL interface definition.
4. The modeling extension creates a new system folder in the Integration Explorer and stores a copy of the WSDL definition file.
5. To integrate a particular service the integration modeler selects the graphical element *new context integration process* from CIP Dashboard Palette, places it on a corresponding layer in Graphical Editor area, and connects it with the new *context provisioning system* component.
6. The modeling extension creates a new integration process's folder in the Integration Explorer, generates a BPEL process file and WSDL interface definition file, and creates a copy of the system's WSDL definition file.
7. The integration modeler selects the *context integration process* in the Graphical Editor area to show and edit its information in Properties View.
8. The integration modeler defines the WSDL interface details and implements the BPEL process using the traditional BPEL modeling perspective.

9. Finally, the integration modeler updates the CIPs repository to make the new context integration processes available for context-aware compositions.

Development of System Independent Aggregation Processes This use case is abstract and defines the aggregation of information obtained from various context provisioning systems. It describes the standard course of actions performed by an integration modeler and specifies the behavior of the modeling extension to support code generation.

Preconditions: The CIPs repository is bound and contains context integration processes for at least two different context provisioning systems. The WSDL interface definitions of the context provisioning systems exist, is validated, and contains no errors. A graphical perspective extended to support development of context integration processes is opened.

Postconditions: The system independent context aggregation processes are defined and uploaded to CIPs repository.

Basic course of actions:

1. The integration modeler selects the graphical element *new System Independent aggregation process* from the CIP Dashboard Palette and places it in the Graphical Editor area.
2. The integration modeler selects a *System Independent aggregation process* element in the Graphical Editor to show and edit its preferences in the Properties View.
3. The integration modeler selects the context integration processes representing interfaces of different context provisioning systems. Subsequently, the integration modeler specifies WSDL interface details to obtain context information.
4. The modeling extension creates a new aggregation's folder in the Integration Explorer, generates a BPEL process file, and WSDL interface definition file, and creates a copy of the WSDL definition files of the involved context integration processes.
5. The modeling extension generates partner links, variables, and invocation logic required for the communication with other context integration processes.
6. The integration modeler implements the BPEL process using traditional BPEL modeling perspective.
7. Finally, the integration modeler updates the CIPs repository to make the aggregation process available for further usage.

Chapter 7

Summary and Future Work

This diploma thesis provided a general discussion about integration and aggregation of context information into service compositions realized in WS-BPEL. Having investigated the work done by others we evaluated the research achievements over the past years in this area and provided the current state of the art of context and techniques of Context-Aware Computing. In Section 2.1.2, we presented a conceptual architecture model for comprehensive context management including sensing, modeling, reasoning, and distributing of context information. After giving necessary background knowledge, we investigated the application of the available techniques of Context-Aware Computing to the domain of service compositions. Due to the fact that context knowledge represents information which is external to service compositions and therefore is managed by external context provisioning systems, we presented the available concepts of integration of context knowledge into service compositions. Finally, in Section 2.3.3, we evaluated some of the existing integration concepts and selected Context Integration Processes as the most suitable integration concept.

Subsequently, the Context Integration Processes concept was used to integrate the C-CAST context management framework (Section 3.3) and Google Maps Web services (Section 4.4.4) into service compositions. In Section 4.3, we presented the extension to the Context Integration Processes concept to support the aggregation of context information obtained from various context provisioning systems. To validate the realized concepts, the composition engines Apache ODE and OW2 Orchestra were used for deployment of the developed context integration processes. The evaluation in Section 5.3 determined that the Context Integration Processes concept fulfills the stated integration objectives. The realized concept enables composition engine independent integration of context, provides encapsulation and outsourcing of context aggregation logic, and represents an extensible approach to integrate different context provisioning systems.

In Chapter 6, we provided the general specification and design details for the WS-BPEL modeling tool extension to support modeling of context-aware service compositions and development of context integration processes. We identified application domains, specified user groups, and described use cases to outline the extension functionality. Finally, to ease future work a guide for setting up the development environment is provided in Appendix section. Additionally, it gives an overview of the C-CAST CMF RESTful Web service interfaces, message formats, and invocation parameters.

Throughout this thesis different areas for further work have been identified. By addressing them, the Context Integration Processes concept can be further improved:

1. **Performance Improvement** The numerous abstraction layers of the Integration Process Pattern had a negative impact on the overall system performance. Optimizations could be applied to reduce the time spent on invocation of context integration processes.
2. **Implementation of context modeling tool** Based on the specification and design details presented in Chapter 6, the extension to a WS-BPEL modeling tool could be implemented to ease the development of context-aware service compositions and context integration processes.
3. **Implementation of CIP management tool** Due to the fact that the proposed context integration approach represents a complex hierarchical Web service stack, there is a demand for a tool which will simplify management of context integration processes, provide monitoring of the integration process' instances, and audit communication between multiple composite flows.
4. **Runtime adaptation** There is a demand to investigate if the dynamic adaptation of business process instances is required. Context knowledge could be used to adjust the control flow of a WS-BPEL instance or exchange partners involved in the interaction with a context-aware service composition.

Appendix A

Appendix

Due to the lack of a complete and high quality documentation about C-CAST CMF interfaces, all available information is gathered in this appendix. The first two sections are devoted to the types of entities and context scopes supported by the current version of C-CAST CMF. The XML Schema data model representing *Context Management Language* (ContextML) vocabulary, context model, and data types can be found on the enclosed CD. The current version of ContextML XML schema is *1.7*. The next section describes the C-CAST CMF RESTful Web service interfaces, message formats and invocation parameters. Finally, the installation and configuration guide is provided in the last section to ease the installation and further development of the context integration processes.

A.1. C-CAST CMF Supported Entities

The current version of C-CAST CMF Context Broker is *1.4.3*. This appendix section illustrates currently supported entity types, shown in Table A.1.

A.2. C-CAST CMF Supported Context Scopes

This section describes the context scopes supported by the current version of the C-CAST CMF Context Broker.

Entity Type	Description	Syntax (type | identifier)
Users	Identifies a user of the C-CAST CMF.	username | <name>
Service Account	Identifies an account for an Internet service associated with a user. The account identifier should be prefixed with the service name code. Sample service name codes include *mailto, facebook, skype, etc.*	serviceAccount | <prefix>:<account_id>
Cellular Terminal	Identifies a mobile device with GSM cellular connectivity.	imei | <imei_id>
Room	Identifies a room in a building, office, meeting room, etc.	room | <roomId>
Personal Computer	Identifies a personal computer equiped with Bluetooth interface for identification.	pc | <BT_addr>
Smart Object	Identifies a smart object which provides some service to users in the environment. A smart object should be equiped with Bluetooth interface for identification.	smartObj | <BT_addr>
Wireless Network Node	Identifies a base connectivity station like a WiFi Access Point or a Cellular Network Base Station identified by WiFi MAC address or Cell Global Identity respectively.	wnn | <node_id>

Table A.1.: C-CAST CMF – Entity Types

A.2.1. Context Scope `userProfile`

The context scope named `userProfile` contains detailed information about user's personal profile. Table A.2 illustrates the related context parameters.

The ContextML `parA` elements encapsulate further user's personal profile data such as emails, work places, mobile phone numbers, Internet messenger service accounts and residential places. Each ContextML `parA` element is named respectively to the enclosed content. The following paragraphs demonstrate the context parameters used to represent each particular `parA` element.

ContextML Element `emailsArray`

Table A.3 contains the information about the user's email addresses structured as a list of ContextML `parS` elements. The corresponding root `parA` element is titled *emails*. Each ContextML `parS` element is named *email* and contains context parameters as listed below.

Parameter Name	XMLSchema data type	Description	Required	Empty
cid	xs:string	Identifier	yes	no
alias	xs:string	User alias	yes	no
firstName	xs:string	User first name	yes	no
lastName	xs:string	User last name	yes	no
gender	xs:string	User gender (m, f)	no	–
birthCity	xs:string	User birth city	no	–
birthCountry	xs:string	User birth country	no	–
birthZip	xs:string	User birth zip code	no	–
birthState	xs:string	User birth state	no	–
birthDate	xs:dateTime	User birth date	no	–
civilStatus	xs:string	User civil status	no	–
fiscalCode	xs:string	User fiscal code	no	–
nationality	xs:string	User nationality	no	–
emails	ContextML parA element	Array of user's email addresses.	yes	yes
works	ContextML parA element	Array of user's work places.	yes	yes
mobiles	ContextML parA element	Array of user's mobile numbers.	yes	yes
instantMessengers	ContextML parA element	Array of user's instant messengers.	yes	yes
homes	ContextML parA element	Array of user's residential places.	yes	yes

Table A.2.: C-CAST CMF – Definition of `userProfile` Scope

Parameter Name	XMLSchema data type	Description	Required	Empty
type	xs:string	Type of email address (personal, business)	yes	no
preferred	xs:boolean	True / False	yes	no
email	xs:string	Email address	yes	no

Table A.3.: C-CAST CMF – Definition of `emails` Element Contained in `userProfile` Scope

ContextML Element `worksArray`

Table A.4 contains the information about the user's work places structured as a list of ContextML `parS` elements. The corresponding root `parA` element is titled *works*. Each ContextML `parS` element is named *work* and contains context parameters as listed below.

Parameter Name	XMLSchema data type	Description	Required	Empty
type	xs:string	Type of user's work place	yes	no
organization	xs:string	User's work organization	yes	no
address	xs:string	Address of user's work place	yes	no
city	xs:string	City of user's work place	yes	no
zip	xs:string	Zip code of user's work place	yes	no
state	xs:string	State of user's work place	yes	no
country	xs:string	Country of user's work place	yes	no
phone	xs:string	Phone number of user's work place	yes	no
fax	xs:string	Fax number of user's work place	yes	no

Table A.4.: C-CAST CMF – Definition of `works` Element Contained in `userProfile` Scope

ContextML Element `mobilesArray`

Table A.5 contains the information about the user's mobile phone numbers structured as a list of ContextML `parS` elements. The corresponding root `parA` element is titled *mobiles*. Each ContextML `parS` element is named *mobile* and contains context parameters as listed below.

Parameter Name	XMLSchema data type	Description	Required	Empty
type	xs:string	Type of mobile number (personal, business)	yes	no
preferred	xs:boolean	True / False	yes	no
mobile	xs:string	Mobile number in the international representation	yes	no
imsi	xs:string	IMSI number	yes	yes

Table A.5.: C-CAST CMF – Definition of `mobiles` Element Contained in `userProfile` Scope

ContextML Element `messengersArray`

Table A.6 contains the information about the user's instant messenger accounts structured as a list of ContextML `parS` elements. The corresponding root `parA` element is titled *messengers*. Each ContextML `parS` element is named *instantMess* and contains context parameters as listed below.

Parameter Name	XMLSchema data type	Description	Required	Empty
type	xs:string	IM account type (personal, business)	yes	no
preferred	xs:boolean	True / False	yes	no
provider	xs:string	IM provider	yes	no
account	xs:string	User's IM account	yes	no

Table A.6.: C-CAST CMF – Definition of instantMess Element Contained in userProfile Scope

ContextML Element homesArray

Table A.7 contains the information about the user's residential places structured as a list of ContextML parS elements. The corresponding root parA element is titled *homes*. Each ContextML parS element is named *home* and contains context parameters as listed below.

Parameter Name	XMLSchema data type	Description	Required	Empty
type	xs:string	Type of user's home	yes	no
address	xs:string	Address of user's home	yes	no
city	xs:string	City of user's home	yes	no
zip	xs:string	Zip code of user's home	yes	no
state	xs:string	State of user's home	yes	no
country	xs:string	Country of user's home	yes	no
phone	xs:string	Phone number of user's home	yes	no
fax	xs:string	Fax number of user's home	yes	no
flat	xs:string	User's home flat	yes	no

Table A.7.: C-CAST CMF – Definition of homes Element Contained in userProfile Scope

A.2.2. Context Scope position

The context information contained in this scope specifies the geographical position of an entity. Table A.8 illustrates the related context parameters.

Possible technologies used for computation of an entity position include: Bluetooth, Wifi, IP-based location, location obtained from GPS sensor, and others.

Parameter Name	XMLSchema data type	Description	Required	Empty
latitude	xs:float	Latitude (WGS84 system)	yes	no
longitude	xs:float	Longitude (WGS84 system)	yes	no
accuracy	xs:int	Position accuracy (meters)	yes	no
locMode	xs:string	Technology used for position computation	yes	no
altitude	xs:float	Altitude	no	–

Table A.8.: C-CAST CMF – Definition of `position` Scope

A.2.3. Context Scope `civilAddress`

The information identifying the home address of an entity is contained within the `civilAddress` scope. Table A.9 illustrates the related context parameters.

Parameter Name	XMLSchema data type	Description	Required	Empty
room	xs:string	Room identifier	no	–
corridor	xs:string	Corridor identifier	no	–
floor	xs:string	Floor	no	–
building	xs:string	Building	no	–
street	xs:string	Street	no	–
postalCode	xs:string	Zip code	no	–
city	xs:string	City	no	–
subdivision	xs:string	State / Area	no	–
country	xs:string	Country	no	–
countryCode	xs:string	Country code in ISO 3166-1 alpha-3 format	no	–
placeType	xs:string	If the civil address refers to a public place this is the place type	no	–
placeName	xs:string	If the civil address refers to a public place this is the place name	no	–

Table A.9.: C-CAST CMF – Definition of `civilAddress` Scope

Parameter Name	XMLSchema data type	Description	Required	Empty
with	xs:string	Type os nearby users (friends, family, colleagues)	no	–
status	xs:string	Status (crowded)	no	–
users	ContextML parA element	Array of nearby users. See Table A.11.	yes	yes

Table A.10.: C-CAST CMF – Definition of userProximity Scope

A.2.4. Context Scope userProximity

The context scope userProximity contains the context information about the entities located in a proximity of a specific entity. Table A.10 illustrates the related context parameters.

Table contains the information about the nearby entities structured as a list of ContextML parS elements. The corresponding root parA element is titled *users*. Each ContextML parS element is named *user* and contains context parameters as listed below.

Parameter Name	XMLSchema data type	Description	Required	Empty
userName	xs:string	Name of the nearby user	yes	no
fullName	xs:string	Full name of the nearby user	no	–
tech	xs:string	Technology used for proximity detection	yes	no
prox	xs:string	Proximity level estimation	no	–
rel	xs:string	Relation between the reference entity and nearby user (friend, family, colleague). If the relation is indirect, the relationship chain is expressed with the separator ;:	no	–
interm	xs:string	If the previous relation is indirect, this parameter contains the username of the intermediary, otherwise it is not present.	no	–

Table A.11.: C-CAST CMF – Definition of users Element Contained in userProximity Scope

A.3. C-CAST CMF RESTful Web Service Interface

This section describes in detail the RESTful Web service API of the C-CAST context management framework. Available interfaces are organized according to the main components of the CMF architecture.

A.3.1. Context Broker Interfaces

The current version of C-CAST CMF Context Broker is *1.4.3*. The following paragraphs illustrate currently exposed interfaces.

Whois Interface

Whois interface provides general information about the current version of the Context Broker.

- **Invocation URL** must be of the following form:

  `HTTP://<HOSTNAME>:<PORT>/CB/ContextBroker/getName`

- **Input parameters** are not defined.
- **Request message** is not defined.
- **Response message** is illustrated in Listing A.1.
- **Fault message** contains an acknowledgement message with the error code and invocation status as shown in Listing A.8.

Listing A.1 ContextML Schema Element for `getName` Response

```
<contextML>
  <ctxResp>
    <contextProvider id="CB" v="1.4.3"/>
    <entity id="" type=""/>
    <scope/>
    <resp status="OK" code="200"/>
  </ctxResp>
</contextML>
```

Listing A.2 ContextML Context Provider Advertisement Schema Element

```
<contextML>
  <ctxAdvs>
    <ctxAdv>
      <contextProvider id="LP" v="1.1.0"/>
      <urlRoot>ca_example.tilab.com/LP</urlRoot>
      <scopes>
        <scopeDef n="position">
          <url>/loc/getLocation</url>
          <entityTypes>username,mobile</entityTypes>
            <inputDef>
              <inputEl name="phone" type="currentSettings:mobile"/>
              <inputEl name="cgi" type="cell:cgi"/>
              <inputEl name="btList" type="bt:btList"/>
              <inputEl name="wfList" type="wf:wfList"/>
            </inputDef>
        </scopeDef>
      </scopes>
    </ctxAdv>
  </ctxAdvs>
</contextML>
```

Context Provider Advertising Interface

The Context Broker provides an interface to register new Context Providers. Each Context Provider informs the Context Broker about its capabilities using an advertisement message encoded in the ContextML format. Entries about registered Context Providers are kept in the Context Provider Lookup table and are linked to an expiry timer. Hence, the Context Provider is responsible to refresh the lookup entries by periodically advertising its capabilities.

- **Invocation URL** must be of the following form:

  `HTTP://<HOSTNAME>:<PORT>/CB/ContextBroker/providerAdvertising`

- **Input parameters** are not defined.
- **Request message** contains the Context Provider advertisement illustrated in Listing A.2. A new Context Provider has to provide the information how to access it (`urlRoot`) along with the invocation parameters. Furthermore, a Context Provider should specify its name, version, entity types, and the supported context scopes. A single advertisement message may contain several context scopes.
- **Response message** contains an acknowledgement message with the registration status and operation code. A sample acknowledgement message is illustrated in the Listing A.7.

- **Fault message** contains an acknowledgement message with the error code and invocation status as shown in Listing A.8.

Available Context Providers Query Interface

The Context Broker exposes the query interface to distribute information about the registered Context Providers. To refine the returned result set the corresponding query should contain information about context scopes. The Context Broker will return the registered Context Providers which support the desired context scopes.

- **Invocation URL** must be of the following form:

  `HTTP://<HOSTNAME>:<PORT>/CB/ContextBroker/getContextProviders`

- **Input parameters** contain `scope` which represents a comma-separated list of context scopes.
- **Request message** is not defined.
- **Response message** contains a list of the available Context Providers and is illustrated in Listing A.3.
- **Fault message** contains an acknowledgement message with the error code and invocation status as shown in Listing A.8.

Listing A.3 ContextML Schema Element for `getContextProviders` Response

```
<contextML>
  <ctxPrvEls>
    <ctxPrvEl>
      <par n="scope">position</par>
      <parA n="contextProviders">
        <parS n="contextProvider">
          <par n="id">LP</par>
          <par n="url">tilab.com/LP/loc/getName</par>
        </parS>
      </parA>
    </ctxPrvEl>
  </ctxPrvEls>
</contextML>
```

Active Entities Query Interface

A list of active entities can be acquired by invoking the corresponding Context Broker interface.

- **Invocation URL** must be of the following form:

  `HTTP://<HOSTNAME>:<PORT>/CB/ContextBroker/getActiveEntities`

- **Input parameters** are not defined.
- **Request message** is not defined.
- **Response message** contains a list of the available active entities as illustrated in Listing A.4.
- **Fault message** contains an acknowledgement message with the error code and invocation status as shown in Listing A.8.

Listing A.4 ContextML Schema Element for `getActiveEntities` Response

```
<contextML>
  <ctxResp>
    <contextProvider id="CB" v="1.4.3"/>
    <timestamp>2011-06-06T16:56:06+02:00</timestamp>
    <method>getActiveEntities</method>
    <resp status="OK" code="200"/>
    <dataPart>
      <parA n="entities">
        <par n="entity">imei|353231005978493</par>
        <par n="entity">imei|353231005978492</par>
        <par n="entity">imei|123456789345</par>
      </parA>
    </dataPart>
  </ctxResp>
</contextML>
```

Context Information Query Interface

The Context Broker exposes a generic interface to distribute context information.

- **Invocation URL** must be of the following form:

  `HTTP://<HOSTNAME>:<PORT>/CB/ContextBroker/getContext?parameters`

- **Input parameters** are separated with the ampersand `&` character and contain:
  1. `entity` specifies the entity type and identifier and should be of the form: `<type>|<identifier>`.
  2. `entities` can be used instead of `entity` parameter in case context information should be acquired for numerous entities. This parameter expects a comma-separated entity list of the form: `<type>|<identifier>`.
  3. `scopeList` represents a comma-separated list of context scopes
- **Request message** is not defined.
- **Response message** contains generated context information as illustrated in Listing A.5.
- **Fault message** contains an acknowledgement message with the error code and invocation status as shown in Listing A.8.

Listing A.5 ContextML Schema Element for `getContext` Response

```
<contextML>
  <ctxEls>
    <ctxEl>
      <contextProvider id="LP" v="1.1.0" />
      <entity type="username" id="Sherlock" />
      <scope>civilAddress</scope>
      <timestamp>2011-02-27T12:20:11+01:00</timestamp>
      <expires>2011-02-27T13:20:11+01:00</expires>
      <dataPart>
        <parS n="civilAddress">
          <par n="street">221 B Baker St</par>
          <par n="city">London</par>
          <par n="country">England</par>
        </parS>
      </dataPart>
    </ctxEl>
  </ctxEls>
</contextML>
```

Context Update Interface

The Context Broker exposes a generic interface to update context information stored in the Context Cache.

- **Invocation URL** must be of the following form:

  `HTTP://<HOSTNAME>:<PORT>/CB/ContextBroker/contextUpdate`

- **Input parameters** are not defined.
- **Request message** contains ContextML element encapsulating the updated context information. A sample context update request is illustrated in the Listing A.6.
- **Response message** contains an acknowledgement message with the update status and operation code. A sample acknowledgement message is illustrated in Listing A.7.
- **Fault message** contains an acknowledgement message with the error code and invocation status as shown in Listing A.8.

Listing A.6 ContextML Schema Element for `contextUpdate` Request

```
<contextML>
  <ctxEls>
    <ctxEl>
      <contextProvider id="LP" v="1.1.0" />
      <entity type="username" id="Harry" />
      <scope>civilAddress</scope>
      <timestamp>2011-02-27T12:20:11+01:00</timestamp>
      <expires>2011-02-27T13:20:11+01:00</expires>
      <dataPart>
        <parS n="civilAddress">
          <par n="room">The Cupboard under the Stairs</par>
          <par n="street">4 Privet Drive</par>
          <par n="city">Little Whinging</par>
          <par n="country">Surrey</par>
        </parS>
      </dataPart>
    </ctxEl>
  </ctxEls>
</contextML>
```

Acknowledgement Message

Context Broker replies with an acknowledgement message to confirm the reception and completeness of the management invocations. An acknowledgement message identifies the component sending the acknowledgement, specifies timestamp and name of the invoked operation. Depending on the invoked method, entity and scope are optionally encoded in the acknowledgement message. Each acknowledgement message contains the invocation status, HTTP status code and other useful information provided in `msg` element. Listings A.7 and A.8 demonstrate sample acknowledgement and fault messages respectively.

Listing A.7 ContextML Acknowledgement Message

```
<contextML>
  <ctxResp>
    <contextProvider id="CB" v="0.1" />
    <entity id="" type="" />
    <scope />
    <method>providerAdvertising</method>
    <resp status="OK" code="200" />
  </ctxResp>
</contextML>
```

Listing A.8 ContextML Fault Message

```
<contextML>
  <ctxResp>
    <contextProvider id="CB" v="1.4.3"/>
    <timestamp>2011-06-07T17:48:27+02:00</timestamp>
    <entity id="12345345" type="imei"/>
    <scope>position</scope>
    <method>getContext</method>
    <resp status="ERROR" code="455" msg="Context Element Not Available: No Context Available
        to Invoke Provider LP"/>
  </ctxResp>
</contextML>
```

A.3.2. Context Provider Interfaces

The context query interface of a Context Provider has the same syntax and semantics as the related interface exposed by the Context Broker. Additionally, the implementation of a specific provider method varies from every other registered Context Provider. Hence, the detailed specification of a Context Provider API should be furnished along with its implementation. The following paragraphs illustrate interfaces exposed by a Context Provider.

Context Information Query Interface

Context Provider exposes a generic interface to distribute context information.

- **Invocation URL** must be of the following form:

  `HTTP://<HOSTNAME>:<PORT>/<PATH>/getContext?parameters`

- **Input parameters** are separated with the ampersand `&` character and expect:

1. `entity` specifies the entity type and identifier and should be of the form: `<type>|<identifier>`.
2. `entities` can be used in place of `entity` parameter in case context information should be acquired for numerous entities. This parameter expects a comma-separated entity list of the form: `<type>|<identifier>`.
3. `scopeList` represents a comma-separated list of context scopes
4. Specific additional parameters defined by every particular Context Provider

- **Request message** is not defined.
- **Response message** contains generated context information and is illustrated in Listing A.5.
- **Fault message** contains an acknowledgement message with the error code and invocation status as shown in Listing A.8.

Generic Context Provider Interface

Context Provider may advertise specific interfaces to query or manipulate context information.

- **Invocation URL** must be of the following form:

  `HTTP://<HOSTNAME>:<PORT>/<PATH>/MethodName?parameters`

- **Input parameters** are optional. They are separated with the ampersand `&` character and represented through:
  1. `parameter name`
  2. `parameter value`
- **Request message** contains ContextML element encapsulating the updated context information. Request message is optional.
- **Response message** contains ContextML element.
- **Fault message** contains an acknowledgement message with the error code and invocation status as shown in Listing A.8.

A.4. Installation and Configuration Guide

This chapter describes the installation steps to make it easier for the reader to get started with the exploitation of Context Integration Processes.

Installing Apache Tomcat

The starting point is to get the Apache Tomcat which can be found on the enclosed CD or downloaded from the Apache Tomcat website [53]. The version of Apache Tomcat used in this diploma thesis is *7.0.19*. After downloading the Apache Tomcat binary, it should be copied to the intended installation directory and expanded.

Configuring the Administrator and Manager Web Interface

To control resources, data sources, users and groups, as well as to perform management tasks on web applications through a simplified web user interface a new user with corresponding roles should be defined. To do this, the reader should add the following lines to the user configuration located in *CATALINA_BASE/conf/tomcat-users.xml* file:

Listing A.9 Configuring the Administrator and Manager Web Interface

```
<role name="admin"/>
<role name="manager-gui"/>

<user name="your_login_name" password="your_login_password" roles="admin,manager-gui"/>
```

Configuring Memory Management

To avoid *Out Of Memory PermGen* errors which are commonly seen during development of context integration processes, the memory available to Apache Tomcat must be increased. To do this, the following line needs to be added to the beginning of the Tomcat startup script file *CATALINA_BASE/bin/catalina.sh*:

Listing A.10 Configuring Memory Management

```
JAVA_OPTS="-Djava.awt.headless=true -Dfile.encoding=UTF-8 -server -Xms1536m -Xmx1536m
    -XX:NewSize=256m -XX:MaxNewSize=256m -XX:PermSize=256m -XX:MaxPermSize=256m
    -XX:+DisableExplicitGC"
```

Installing Apache ODE

The Apache ODE binary can be found on the enclosed CD or downloaded from the Apache ODE website [58]. The version of Apache ODE used in this diploma thesis is *1.3.5*. After successful download of the Apache ODE binary, it should be expanded. Finally, to deploy Apache ODE the Web application archive file called *ode.war* is placed into the Tomcat *CATALINA_BASE/webapps/* folder.

Installing OW2 Orchestra

The OW2 Orchestra binary file is located on the enclosed CD. Alternatively, the reader can download OW2 Orchestra binaries from OW2 Orchestra website [59]. The version of Orchestra engine used in this diploma thesis is *4.8.0*. To install OW2 Orchestra the reader should expand the archive and rename the following files located in *ORCHESTRA_BASE/lib* directory:

- *console-4.8.x.war* to *console.war*
- *orchestra-cxf-war-4.8.x.war* to *orchestra.war*
- *orchestradesigner-4.8.x.war* to *designer.war*

Finally, the renamed files should be copied to the Tomcat *CATALINA_BASE/webapps/* folder to deploy OW2 Orchestra on Apache Tomcat.

Starting / Stopping Apache Tomcat

To start or to stop Apache Tomcat the reader should proceed to the directory *CATALINA_BASE/bin* and execute *startup.sh (startup.bat)* or *shutdown.sh (shutdown.bat)* correspondingly.

Installing C-CAST CMF SOAP WSDL Interface

The next step is to deploy the SOAP WSDL interface developed in Section 3.3 for C-CAST CMF Context Broker and Context Provider services. To do this, the corresponding Web application archive file *CMF.war* should be copied from the enclosed CD to the Tomcat *CATALINA_BASE/webapps/* folder. Finally, the reader can optionally test the deployed Web application using tests defined for this component to ensure that the C-CAST CMF SOAP Web service interface was successfully deployed.

Installing Google Maps SOAP WSDL Interface

In the same way, the SOAP WSDL interface developed for Google Maps Web services in Section 4.4.4 should be deployed on Apache Tomcat. The corresponding Web application archive file *GoogleMapsServices.war* should be copied from the enclosed CD to the Tomcat *CATALINA_BASE/webapps/* folder.

Installing Context Integration Processes

To complete the installation, the reader should install the context integration processes developed to integrate C-CAST CMF and Google Maps Web services. To do this, the archive files located on the enclosed CD should be deployed using web user interfaces exposed by Orchestra and Apache ODE. Finally, the reader can optionally test the deployed integration processes using tests defined for each particular context integration process or by deploying the sample TaxiServiceProvider service composition.

Bibliography

[1] National Institute of Standards and Technology (NIST). The NIST Definition of Cloud Computing. [Online]. Available: http://csrc.nist.gov/publications/drafts/800-145/Draft-SP-800-145_cloud-definition.pdf (Cited on page 1)

[2] OASIS Standard. Web Services Business Process Execution Language. [Online]. Available: http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html (Cited on pages 1, 44 and 78)

[3] 4CaaSt. PaaS Cloud Platform Project. [Online]. Available: http://4caast.morfeo-project.org/ (Cited on page 1)

[4] Oxford English Dictionary Online. Oxford University Press. [Online]. Available: http://www.oed.com/view/Entry/40207 (Cited on page 8)

[5] C. Bolchini, C. A. Curino, E. Quintarelli, F. A. Schreiber, and L. Tanca, "A Data-oriented Survey of Context Models," *SIGMOD Rec.*, vol. 36, December 2007. (Cited on pages 8 and 17)

[6] R. Want, A. Hopper, V. Falcão, and J. Gibbons, "The Active Badge Location System," *ACM Trans. Inf. Syst.*, vol. 10, pp. 91–102, January 1992. (Cited on page 8)

[7] M. Weiser, "The Computer for the 21st Century," *Scientific American*, February 1991. [Online]. Available: http://www.ubiq.com/hypertext/weiser/SciAmDraft3.html (Cited on page 8)

[8] B. Schilit and M. Theimer, "Disseminating Active Map Information to Mobile Hosts," *Network, IEEE*, vol. 8, no. 5, September 1994. (Cited on pages 9 and 11)

[9] B. Schilit, N. Adams, and R. Want, "Context-Aware Computing Applications," *Mobile Computing Systems and Applications, IEEE Workshop on*, 1994. (Cited on page 9)

[10] M. Kaenampornpan and E. O'Neill, "An Integrated Context Model: Bringing Activity to Context," in *Workshop on advanced context modelling, reasoning and management - UBICOMP*. (Cited on pages 9 and 10)

[11] G. D. Abowd and A. K. Dey, "Towards a Better Understanding of Context and Context-Awareness," in *Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*. Springer-Verlag, 1999. (Cited on page 9)

[12] A. Soylu, P. Causmaecker, and P. Desmet, "Context and Adaptivity in Pervasive Computing Environments: Links with Software Engineering and Ontological Engineering," *Journal of Software*, vol. 4, no. 9, 2009. (Cited on pages 10, 13, 15, 16, 17, 18 and 53)

[13] A. K. Dey, "Understanding and Using Context," *Personal and Ubiquitous Computing*, vol. 5, 2001. (Cited on pages 10, 11 and 18)

[14] G. D. Abowd and E. D. Mynatt, "Charting Past, Present, and Future Research in Ubiquitous Computing," *ACM Transactions on Computer-Human Interaction*, vol. 7, March 2000. (Cited on page 10)

[15] B. Schilit, N. Adams, and R. Want, "Context-Aware Computing Applications," in *Proceedings of the 1994 First Workshop on Mobile Computing Systems and Applications*. Washington, DC, USA: IEEE Computer Society, 1994. (Cited on page 10)

[16] P. Prekop and M. Burnett, "Activities, Context and Ubiquitous Computing," *CoRR*, vol. cs.IR/0209021, 2002. (Cited on page 10)

[17] T. Hofer, W. Schwinger, M. Pichler, G. Leonhartsberger, J. Altmann, and W. Retschitzegger, "Context-Awareness on Mobile Devices – the Hydrogen Approach," in *Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03)*. Washington, DC, USA: IEEE Computer Society, 2003. (Cited on page 10)

[18] M. Baldauf, S. Dustdar, and F. Rosenberg, "A Survey on Context-aware Systems," *Int. J. Ad Hoc Ubiquitous Comput.*, vol. 2, June 2007. (Cited on pages 10 and 17)

[19] G. Chen and D. Kotz, "A Survey of Context-Aware Mobile Computing Research," Hanover, NH, USA, Tech. Rep., 2000. (Cited on pages 10 and 18)

[20] R. Hull, P. Neaves, and J. Bedford-Roberts, "Towards situated computing," in *Proceedings of the 1st IEEE International Symposium on Wearable Computers*. Washington, DC, USA: IEEE Computer Society, 1997. (Cited on page 11)

[21] M. J. Pascoe, "Adding Generic Contextual Capabilities to Wearable Computers," in *Proceedings of the 2nd IEEE International Symposium on Wearable Computers*. Washington, DC, USA: IEEE Computer Society, 1998. (Cited on page 11)

[22] M. Mrissa, C. Ghedira, D. Benslimane, Z. Maamar, and U. C. B. L. Villeurbanne, "A Context Model for Semantic Mediation in Web Services Composition," in *In Proceedings of the 25th International Conference on Conceptual Modeling*. Springer. (Cited on page 12)

[23] C. Ghedira and H. Mezni, "Through Personalized Web Service Composition Specification: From BPEL to C-BPEL," *Electronic Notes in Theoretical Computer Science*, vol. 146, no. 1, 2006, proceedings of the First International Workshop on Context for Web Services (CWS 2005). (Cited on page 12)

[24] A. A. George and P. A. S. Ward, "An architecture for providing context in WS-BPEL processes," in *Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds*, 2008. (Cited on pages 12, 23 and 24)

[25] L. Li, D. Liu, and A. Bouguettaya, "Semantic Based Aspect-oriented Programming for Context-aware Web Service Composition," *Information Systems*, vol. 36, no. 3, 2011. (Cited on page 13)

[26] Quan, "Techniques on Developing Context-aware Web Services," *International Journal of Web Information Systems*, vol. 6, no. 3, 2010. (Cited on page 13)

[27] A. K. Dey, G. D. Abowd, and D. Salber, "A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-aware Applications," *Hum.-Comput. Interact.*, vol. 16, December 2001. (Cited on page 13)

[28] J. Indulska and P. Sutton, "Location Management in Pervasive Systems," in *Proceedings of the Australasian information security workshop conference on ACSW frontiers 2003 - Volume 21*. Australian Computer Society, Inc., 2003. (Cited on page 15)

[29] T. Strang and C. L. Popien, "A Context Modeling Survey," in *Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing*, 2004. (Cited on page 15)

[30] J. Ye, L. Coyle, S. Dobson, and P. Nixon, "Ontology-based Models in Pervasive Computing Systems," *Knowl. Eng. Rev.*, vol. 22, December 2007. (Cited on page 16)

[31] X. H. Wang, D. Q. Zhang, T. Gu, and H. K. Pung, "Ontology Based Context Modeling and Reasoning using OWL," in *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*. Washington, DC, USA: IEEE Computer Society, 2004. (Cited on page 16)

[32] P. Moore, B. Hu, X. Zhu, W. Campbell, and M. Ratcliffe, "A Survey of Context Modeling for Pervasive Cooperative Learning," in *Information Technologies and Applications in Education, 2007. ISITAE '07. First IEEE International Symposium on*, November 2007. (Cited on page 16)

[33] H. Chen, "An Intelligent Broker Architecture for Pervasive Context-Aware Systems," Ph.D. dissertation, University of Maryland, Baltimore County, December 2004. (Cited on page 19)

[34] University of Stuttgart. SFB 627: Nexus, Spatial World Models for Mobile Context-Aware Applications. [Online]. Available: http://www.nexus.uni-stuttgart.de/index.en.html (Cited on page 20)

[35] F. Dürr, N. Hönle, D. Nicklas, C. Becker, and K. Rothermel, "Nexus–A Platform for Context-Aware Applications," in *1. GI/ITG Fachgespräch Ortsbezogene Anwendungen und Dienste.*, J. Roth, Ed. Hagen: FernUniversität in Hagen, June 2004. (Cited on pages 20 and 21)

[36] C. R. Becker, *System Support for Context-Aware Computing*, ser. Habilitationsschrift. Stuttgart: Fakultät Informatik, Elektrotechnik und Informationstechnik, Universität Stuttgart, November 2004. (Cited on page 20)

[37] C-CAST. Context Casting project. [Online]. Available: http://www.ict-ccast.eu/ (Cited on page 22)

[38] M. Knappmeyer, S. L. Kiani, C. Frà, B. Moltchanov, and N. Baker, "ContextML: a light-weight context representation and context management schema," in *Proceedings of the 5th IEEE international conference on Wireless pervasive computing*. Piscataway, NJ, USA: IEEE Press, 2010. (Cited on pages 22, 37 and 40)

[39] M. Wieland, P. Kaczmarczyk, and D. Nicklas, "Context Integration for Smart Workflows," in *Proceedings of the 2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications*. Washington, DC, USA: IEEE Computer Society, 2008. (Cited on page 25)

[40] M. Wieland, D. Nicklas, and F. Leymann, "Managing Technical Processes Using Smart Workflows," in *Proceedings of the 1st European Conference on Towards a Service-Based Internet*. Berlin, Heidelberg: Springer-Verlag, 2008. (Cited on page 26)

[41] G. Hermosillo, L. Seinturier, and L. Duchien, "Creating Context-Adaptive Business Processes," in *The 8th International Conference on Service Oriented Computing*, vol. 6470, San Francisco, California États-Unis, 12 2010. (Cited on page 27)

[42] PetalsLink, Petals ESB. Easy BPEL: BPEL Engine. [Online]. Available: http://easybpel.petalslink.org/ (Cited on page 28)

[43] (1999, November) XML Path Language (XPath) Version 1.0. [Online]. Available: http://www.w3.org/TR/xpath (Cited on pages 28 and 52)

[44] Q. Z. Sheng, J. Yu, A. Segev, and K. Liao, "Techniques on Developing Context-aware Web Services," *IJWIS*, vol. 6, no. 3, 2010. (Cited on pages 28 and 29)

[45] The University of Adelaide. ContextServ Project: A Platform for Rapid and Flexible Development of Context-Aware Web Services. [Online]. Available: http://cs.adelaide.edu.au/~contextserv/ (Cited on page 28)

[46] Tigris.org. ArgoUML, the Open Source UML Modeling Tool. [Online]. Available: Tigris.org (Cited on page 30)

[47] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. (2001, March) Web Services Description Language (WSDL) Version 1.1. [Online]. Available: http://www.w3.org/TR/wsdl (Cited on page 44)

[48] JSR 224. Java API for XML-Based Web Services (JAX-WS) 2.0. [Online]. Available: http://jcp.org/en/jsr/detail?id=224 (Cited on pages 44 and 59)

[49] Apache Software Foundation. Apache CXF. [Online]. Available: http://cxf.apache.org/ (Cited on pages 44 and 59)

[50] IBM, Dennis Sosnoski. Java Web Services: CXF Performance Comparison. [Online]. Available: http://public.dhe.ibm.com/software/dw/java/j-jws14-pdf.pdf (Cited on page 44)

[51] JSR 311. JAX-RS: The Java API for RESTful Web Services. [Online]. Available: http://jcp.org/en/jsr/detail?id=311 (Cited on pages 44 and 59)

[52] Eclipse Foundation. Eclipse Platform. [Online]. Available: http://www.eclipse.org/org/ (Cited on pages 44, 59 and 77)

[53] Apache Software Foundation. Apache Tomcat. [Online]. Available: http://tomcat.apache.org/ (Cited on pages 44 and 106)

[54] XSL Transformations (XSLT) Version 1.0. [Online]. Available: http://www.w3.org/TR/xslt (Cited on pages 49 and 52)

[55] Extensible Markup Language (XML) 1.0 (Fifth Edition). [Online]. Available: http://www.w3.org/TR/xml (Cited on pages 52 and 69)

[56] (2011, April) Google Maps/Google Earth APIs Terms of Service. [Online]. Available: http://code.google.com/apis/maps/terms.html#section_10_12 (Cited on page 56)

[57] Google Maps API Premier. [Online]. Available: http://code.google.com/intl/en/apis/maps/documentation/premier/ (Cited on page 56)

[58] Apache Software Foundation. Apache ODE (Orchestration Director Engine). [Online]. Available: http://ode.apache.org/ (Cited on pages 65, 67 and 107)

[59] OW2. OW2 Orchestra Engine. [Online]. Available: http://orchestra.ow2.org/ (Cited on pages 65, 67 and 107)

[60] SmartBear Software. soapUI Web Service Testware. [Online]. Available: http://www.soapui.org/ (Cited on page 68)

[61] Eclipse Foundation. BPEL Designer Project. [Online]. Available: http://www.eclipse.org/bpel/ (Cited on pages 73, 77 and 78)

[62] ——. Eclipse Documentation. [Online]. Available: http://www.eclipse.org/documentation/ (Cited on page 77)

[63] ——. Eclipse Modeling Framework Project. [Online]. Available: http://www.eclipse.org/modeling/emf/ (Cited on page 77)

[64] ——. Graphical Editing Framework Project. [Online]. Available: http://www.eclipse.org/gef/ (Cited on page 78)

[65] B. Höhensteiger, M. Illiger, and S. Moser, "Extending the Eclipse BPEL Designer with Custom Activities," October 2008. [Online]. Available: http://www.eclipse.org/bpel/users/pdf/CreateAnExtensionActivity.pdf (Cited on page 81)

All links were last followed on Juli 30, 2011.

Declaration

All the work contained within this thesis, except where otherwise acknowledged, was solely the effort of the author. At no stage was any collaboration entered into with any other party.

(Rodion Hagin)