INCREMENTAL QUERY ANSWERING OVER SEMANTIC CONTEXTUAL INFORMATION

MOHAMMAD OLIYA

(M.Sc.), NUS

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

NATIONAL UNIVERSITY OF SINGAPORE

2012

CONTENTS

Summary

In the vision of ubiquitous computing, users should get the right information, at the right time, at the right situation. Such provision of appropriate information will assist users in performing their daily tasks in a natural and transparent way. *Context-aware* systems are more flexible, adaptable, and autonomous.

Formal representation of context information is gaining an ever increasing number of advocates in the literature. It fosters interoperability among heterogeneous context sources and eases the development of context-aware applications. Thanks to the representation and reasoning power of their underlying logics, it is also possible to describe complex context data, share and integrate context between heterogeneous entities, deduce abstract or hidden knowledge, and deal with the inconsistency of the data. The Web Ontology Language (OWL) is the standard way for representing the semantics of information in the web, and is the main formal and practical method for modeling context.

One major issue with regards to the application of OWL is the overhead of query answering when changes occur in the observed facts. Traditionally, reasoning on an updated knowledge base is performed from the scratch. As the query answering mechanisms are based on available reasoning techniques, this also results in the re-evaluation of the query from the beginning.

In this dissertation, a novel incremental query answering technique for semantic (OWL-based) contextual information is proposed. The aim is to avoid redundant computations and alleviate the cost of reasoning from scratch. Our method can be applied to the fragments of OWL which can be axiomatized as a set of rules, including Resource Description Format Schema (RDFS) and Description Horn Logics. We consider *instance retrieval queries* which ask for instances of the contextual situations predefined in the ontology. In addition, we only consider changes in the facts (ABox) and not the

changes in the definition of knowledge structure (TBox).

Our technique consists of first translating the ontology schema as well as queries into rules. By targeting the Description Horn Logic fragment of OWL, we are able to represent the ontology schema as a set of definite Horn rules, i.e. rules with only one literal at the head. These rules are then used to build the *Rete* network which incrementally maintains the query results as changes occur in the observed data. As the evaluation of the rules can be computationally expensive, we further introduce an optimization to prune the rules which do not affect query results. The empirical results suggest the practicality of our method for the perceived application domains.

To the best of our knowledge, we are the first to address the problem for context aware systems. The novelty of our method lies in the identification of the proper tools and language fragments that can work in tandem for the expected results. Our method does not need alteration of existing OWL-based reasoners, enabling context aware systems to achieve incremental query answering functionality with minimal changes. In addition, our method supports *hybrid* inference method where application-specific rules can be used in conjunction with pure OWL based reasoning.

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

"The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it." This is how Mark Weiser describes the vision of ubiquitous computing in his seminal paper [3]. Such a utopia is saturated with *pervasive* computing and communication technologies. Everyday objects and places are smart, sensors and actuators maintain rich connections between the physical and virtual worlds, and computing is spread throughout the environment [4]. A highly anticipated property of this new paradigm is its graceful integration with human lives. To be minimally intrusive, a ubiquitous computing environment must be *context-aware*. Contextual information can be used for adapting user interface, tailoring the set of application-relevant data, increasing the precision of information retrieval, discovering services, or making the user interaction implicit [5].

Despite the maturity of required technologies, this vision is still far from reality. The major challenge has been attributed to the *seamless integration* of the component technologies [6]. The sheer diversity of the context sources as well as different soft-

ware which collect, process, and change the context information results in heterogeneity which makes interoperability challenging. The ubiquitous computing community increasingly understands benefits of formal context information modeling. A well-defined semantics alleviates the heterogeneity of context sources and fosters reuse and sharing of information which can be expensive to gather, evaluate and maintain. In fact, it reduces the complexity of context-aware applications and simplifies their development and maintenance. Moreover, a well-defined semantics also enables a number of reasoning services which can be used to derive implicit or abstract information, answer queries, and detect the inconsistencies peculiar to context data [7][1].

Proposed *formal* context models usually take advantage of the well-known standards such as the Web Ontology Language (OWL) [8]. This language is characterized by the formal semantics of Description Logics (DL) [9] and its RDF-based serializations [10]. However, the main limitation in using OWL as the underlying representation model is in the overhead of query answering over under changing data. Traditionally, reasoning on an updated Knowledge Base (KB) is performed from scratch. As the query answering mechanisms are based on available reasoning techniques, this also results in the re-evaluation of the query from the beginning [11]. This problem makes the use of the semantic models impractical due to the efficiency and scalability issues. Nonetheless, none of the proposed work in the literature consider the issue, and readily rely on the inappropriate tools without any reservation. This problem has been considered recently in the semantic web community [11] [12]. The proposed methods, however, are overly complex or limited to theoretical studies. In fact, none of the existing OWL and DL reasoners support incremental query-answering functionality.

The problem I tackle in this dissertation is the incremental query answering for se-

[1]The inconsistency in context data can stem from contradictory information reported by different sources. For example, while camera may identify the location of a user to be in room 1, the indoor positioning system may report his location to be in room 2. The imperfection and uncertainty of context data are further elaborated in section 2

mantic contextual information. The method is to alleviate the cost of reasoning from scratch by reusing the results of previous computations. We rely on the observation that the Rete algorithm [13] - used for the evaluation of rules in expert systems - provides sufficient infrastructure for incremental reasoning. Therefore, we target a limited, yet expressive, subset of the OWL which can be axiomatized as a set of rules. This allows transforming the original OWL query answering problem into rule-based reasoning and therefore leveraging the incremental Rete algorithm.

The Rete algorithm performs inference in a forward chaining manner, which can be costly in terms of computation run time. To alleviate the issue, we further develop an optimization where the set of rules to maintain are adaptively selected based on their effect on the query results. We formally define this property in section 3.3. Given a query, all the rules corresponding to the ontology schema are recursively checked and pruned if they do not influence the query results. The complexity of the optimization is linear in size of the rule base, and thus the size of the ontology schema (Tbox). As we show in our experiments, it can reduce the query answering time significantly, compared with the default Rete implementation.

To the best of our knowledge, we are the first to tackle the problem in the ubiquitous computing community. We highlight the contributions of the work as follows. (i) Proposing a novel approach for incremental reasoning and query answering by identifying a proper subset of OWL and using existing algorithms. (ii) The proposed method requires no significant change in the implementation of existing tools, as we rely on the available Rete-based rule engines. This provides the context aware systems to address the problem with minimal changes. (iii) Our proposed method allows a *hybrid* inference method when the rules resulting from the translation can be augmented with further *application specific rules*. This is specifically important as rule-based reasoning can complement the limitations of pure OWL based reasoning [7][14][15]. More

discussion regarding these arguments are presented in sections 3.1 and 3.4.

Like any other piece of research, the work presented here is based on certain assumptions. For this proposal to be applicable, we assume that the contextual information is expressed in the Description Horn Logic fragment of the Web Ontology Language. Former studies have shown that this fragment is expressive enough for many of the application scenarios [16]. In addition, the changes made to the knowledge base are to be in the assertional level (ABox) and not about the knowledge structure (TBox). Lastly, we study long-running conjunctive retrieval queries which are interested to receive updates for query results as changes occur in the observed facts. A formal definition of this category of the queries is presented in section 3.

The dissertation is organized as follows. In chapter 2, I elaborate on our definition of context and its properties. An argument in favor of semantic models is presented and the rest of chapter discusses their application. In chapter 3, I focus on the query answering problem with regards to the dynamic context information which are expressed in OWL. I detail the problem, assumptions, and the method in this chapter. In the end, we assess the performance of our method on a benchmark where we expect that such an incremental solution should deliver acceptable results in practice. I present the related work in chapter 4 where I discuss how this work stands in relation to existing work. The future research directions are presented in chapter 5, and the dissertation is concluded in chapter 6.

CHAPTER 2

SEMANTIC CONTEXT MODELS

Context aware computing is a new paradigm which has recently become the focus of attention, due to the widespread use of mobile devices and advances in ubiquitous technologies. The overall goal is to make systems and applications more intelligent by personalizing their behavior based on the context. The more context-aware a system is, the more usable and acceptable it will be for human users. Abowd et. al. [17] define the context to be:

> "Any information that can be used to characterize the situation of entities (i.e., whether a person, place or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves."

Useful context information can be retrieved from physical or virtual sensors. Physical sensors refer to hardware sensing devices capturing a wide range of data types. These include location, motion and acceleration, audio, light, visual context (camera images),

touch, temperature, ECG sensors and so on. These sources of information can be consumed directly or be used in deriving more abstract information such as the activity of the users. On the other hand, virtual sensors refer to information sources which are provided by various software applications and services through different APIs. Consider, for example, traffic and weather information, identity, profile, and calendar of users, and flight information.

The choice of a context model is closely related to the performance, flexibility and scalability of context aware systems. The context model should capture the properties of the context and facilitate development of context aware systems. In the rest of this section, I will provide a discussion on the properties of context and the resulting implications for a good model. For in-depth reviews on context properties and existing modeling approaches please refer to survey [18].

Distribution and Heterogeneity

The context is usually derived from a number of heterogeneous data sources and it needs to be filtered, aggregated, and abstracted before use. Each of these data sources may have different data type, representation, and specifications. In fact, context aware systems are distributed and without a central instance responsible for creation, deployment, and maintenance of data and services. The sheer diversity of the context sources as well as different entities which collect, process, and change the context information, will pose serious interoperability issues. In fact, this concern has been attributed as the main problem for development of context aware systems, despite the availability of constituent technologies [4].

The ideal context model should be able to provide seamless interoperability between the context sources and the entities involved. In order to ease the aggregation of the data, powerful knowledge representation methods should be taken. Ad-hoc formalisms with insufficient expressivity and formalism make this process difficult [19]. A formal model

enables knowledge sharing among context sources and the inter-operating systems.

Reasoning

Different application scenarios require data in varying levels of abstraction. For instance, while one may be interested in the location of people in a building, another may be interested to know if a *Meeting* is going on. Generally, low-level data are harder to process, interpret, and act upon. To bridge the gap, we need to abstract low-level context data to more abstract representations through reasoning. The inferred high-level contexts give meaning to a set of lower level contexts, are more stable, and are easier to define and maintain than low level context. Also, more abstract contexts are less susceptible to change and adapting to them is easier for applications [7]. In addition, the high level representation of context will simplify querying for applications as less details needs to be specified (e.g. exact sources of the data). In fact, sophistication of supported queries characterizes the difference between middlewares for context aware systems and sensor networks. Sensor networks usually provide means for trivial operators such as computing for min, max, and average. Context aware systems, on the other hand provide richer and more abstract queries, which implies more complex processing [20].

Imperfection, Uncertainty, and Incompleteness

Context can be *imperfect* in an undesired way, due to hardware problems or non-optimal user behavior. In [21], the authors highlight four types of imperfect context information. Context can be *unknown* when we have no information (e.g. a faulty sensor); *ambiguous*, when we receive conflicting reports (e.g. two different positions reported for a single person, from RFID and camera); *imprecise*, when the data is correct but not exact enough (e.g. reporting the location of a person to be in a region and no exact location are available); and *erroneous*, when the received data are considered to be false. Hence, the model should allow for specifying the quality of context. In this way, applications will be able to work around the issue, e.g. by discarding the retrieved

data or looking for alternative sources of context information.

Moreover, the complexities of contextual interrelationships make any modeling effort error prone [22]. In a formal setting, it is possible to describe context in terms of other context or domain definitions. This will ease the verification of any inconsistency in the knowledge about the context.

Performance

In a real world setting, a context aware system will comprise numerous sensors, users accessing the system, and uncountable information available through other resources such as external databases and the web. We may need to persist context data to perform temporal reasoning, predict future context, or recommend services. Efficient organization, manipulation, and provision of context data are crucial for scalability. Furthermore, context is susceptible to frequent changes (e.g. location, temperature, and road traffic data). The processing and delivery of context data should be efficient and timely, so that the applications are not delivered stale data.

Many context-aware systems using various context models have been developed over the years. For a more comprehensive and detailed study of different context models, interested reader may refer to surveys [7] [23][18][20].

Models based on Key-value pairs use simple dictionary like structures for representation of context. The key can represent the temperature context and the value part, the current reading of a sensor. Markup scheme models usually extend the XML based models such as Composite Capabilities/ Preferences Profile (CC/PP)[1] . Object oriented methods introduce concepts such as encapsulation, inheritance, and reusability into the modeling process. The details of context processing are hidden within objects and access to the context data is provided through specified interfaces. Spatial models stress the significance of location information and provide constructs as well as data storage

[1]A CC/PP profile is a description of device capabilities and user preferences which can be used to guide the adaptation of content presented to that device

and retrieval mechanisms for efficient manipulation of the spatial context. In a logic based context model, the context is defined as facts, expressions and rules. It is added to, updated in, and deleted from a logic based system in terms of facts.

The experiences with the proposed context models and the challenges faced in practice have influenced the set of the requirements defined for context modeling and reasoning. In fact, these modeling approaches have been shown to be insufficient for (i) representing a variety of context types, (ii) capturing relationships, dependencies, and quality of context information, (iii) allowing consistency checking, or (iv) supporting reasoning on context [7].

The pervasive computing community increasingly understands benefits of formal context information modeling [7]. In fact, in order to facilitate interoperability, powerful knowledge representation methods should be taken. Ad-hoc formalisms with insufficient expressivity make this process difficult [19]. Another observation is that the complexities of contextual interrelationships make any modeling effort error prone [22]. As the data is mainly originating from various sources, the chances to encounter inconsistencies are not negligible. In a formal setting, it is possible to verify the existing information against inconsistencies. Formal representation of the context data also facilitates automated reasoning based on the semantics of the data or additional information in the form of rules. This can be helpful in improving the quality of the existing data, inferring new knowledge (e.g. forward chaining) or answering queries (e.g. backward chaining); which are not possible using traditional syntactic approaches.

In this dissertation I consider using ontologies for modeling context, either in purely ontology based or hybrid models. In the next chapter we elaborate on the use of semantic languages for modeling context.

2.1 Using the Web Ontology Language

Ontology is a term originating from philosophy that refers to the science of describing the entities in the world and how they are related. Ontologies have been adopted in several fields of computer science such as information integration, semantic web, natural language processing, and databases [9]. According to W3C, ontologies "define the terms used to describe and present an area of knowledge." They are used to define the semantics of a group of entities and the relationships among them. They intend to provide a shared consistent understanding of a domain, in order to foster knowledge reuse, sharing, and interoperability.

The Web Ontology Language (OWL) is the outcome of the efforts over the past decade to standardize languages for formally representing the semantics of information on the Web; this work has been driven by both the academic and industrial research communities, as well as through initiatives within standardization organizations. OWL expresses information in terms of the Resource Description Format (RDF) [24], which is a standard model for interchangeable data on the Web. Furthermore, OWL is based on Description Logics (DL) [9] - which is widely accepted to provide a solid foundation for ontology representation and reasoning. In fact, the expression of ontologies using a logical language will enable detailed, accurate, consistent, sound, and meaningful distinction of the classes, properties, and relations [25].

Description Logics are expressive decidable fragments of First Order Logics. The basic elements of DL (and thus OWL) are so-called *classes* which group objects into categories, and *properties* which relate pairs of objects with each other. An arbitrary class and property Description can be constructed from two disjoint sets of symbols, Class Names and Property Names (also called Atomic Classes and Atomic Properties) using a variety of class- and property-forming constructors, the range of which is specific for the particular description logic (please refer to [9] for the detailed specification.)

[16].

For instance, we can describe the people all of whose subscriptions are with SingTel as:

$$\text{User} \sqcap \forall \text{ hasSubscription.SingTel}$$

A DL (and thus OWL) *knowledge base* usually consists of a set of axioms, which can be distinguished into terminological axioms (building the so-called TBox $\mathcal{T}$) and assertional axioms or assertions (constituting the ABox $\mathcal{A}$). In analogy to the database systems, TBox corresponds to the database schema while the ABox represents the database instance. The following is a brief description of the parts; for an in-depth review of description logics and its formal specification, please refer to [9].

TBox A TBox is constituted by a finite set of terminological axioms which define subsumption and equivalence relations on classes and properties. An equivalence axiom whose left-hand side is an atomic class (property) is called a *class (property) definition*. The respective class on the left-hand side of the equivalence axiom is called *defined* Class. Axioms of the form C $\sqsubseteq$ D for complex class descriptions C and D are called *(general) inclusion axioms*. C is called a *primitive* class, if it is atomic and occurs on the left-hand side of an inclusion axiom. Moreover, the set of axioms of the form $R \subseteq S$ where both R and S are atomic classes (properties) is called a *class (property) hierarchy*. We say that a class A *directly uses* a class B in $\mathcal{T}$ if B appears on the right-hand side of the definition of A.

ABox Assertional axioms or *Assertions* introduce *Individuals*, i.e. instances of a class, into the knowledge base and relate individuals with each other and the introduced terminology. We can distinguish two kinds of assertions. *Class Assertions* express that an individual is member of a class. *Property assertions* express, that two individuals are related with each other via a given property. For instance, the expression *John:Person* means that *John* is an *individual* which belongs to the *class Person*.

2.1.1 Reasoning

The expression of context data using OWL allows various Description Logics reasoning tasks to be considered. They enable us to draw new conclusions about the knowledge base, check its consistency, or answer queries. For each part of the knowledge base, TBox or ABox, different set of reasoning methods are available.

The TBox reasoning services are with regards to the knowledge structure. They are used in the knowledge engineering phase to assist the domain experts in devising a *consistent* knowledge structure. In addition, they are used to speed up computations with regard to *individuals* (instances) in the $\mathcal{A}$. Please refer to [9] for the detailed specification.

ABox reasoning is about the actual individuals (instances), and how they can be associated to the classes. As the ABox contains class assertions *C(i)* and property assertions *R(a, b)*, the most notable problem is *retrieval*. Given an ABox $\mathcal{A}$ and a class *C*, the purpose is to find all named classes C for an individual *a* for which $\mathcal{A} \models C(a)$. In context aware systems, this can be used to determine the most specific *situation* based on a set of observations. For example, [19][26] and [27] leverage on this form of reasoning, where contextual situations are defined as classes in DL, and the set of sensor inputs are considered to be *individuals* (instances) in the knowledge base. Determining the current context is then formulated as finding the most specific class in the knowledge base to which the *individuals* representing the current context belong to.

The retrieval problem can be dually defined as finding all individuals *a* such that $\mathcal{A} \models C(a)$ [16]. Please note that this task is conceptually different from query answering in Relational Databases. That is, the fact *C(a)* may not be readily available in the knowledge base, but can be *inferred* from the existing set of information.

The ABox retrieval tasks also consider *properties*. Given a property *R* and an individual *i* , the tasks is to retrieve all individuals *x* which are related with *i* via *R*; i.e.

$x|(T, A) \models R(i, x)$. Similarly we can retrieve the set of all named properties *R* between two individuals *i* and *j*, ask whether the pair *(i, j)* is a filler of *P* or ask for all pairs *(i, j)* that are a filler of *P* [16].

Most of the existing DL based reasoners (e.g. Pellet [28] and Racer [29]) are based on the tableau process for detecting inconsistencies. A tableau is a graph corresponding to the underlying model of the knowledge base, with nodes representing individuals and edges representing the relationship between individuals. The algorithm starts with a single individual that satisfies the model and tries to construct a tableau or some structure from which a tableau can be created. Inference proceeds by applying several expansion rules and terminates either when no more rules can be applied, or when contradictions are detected.

The tableau decision procedure theoretically determines the consistency of a set of axioms. Both TBox and ABox reasoning tasks mentioned above can be reduced to the consistency checking problem; therefore, existing DL reasoners mainly use the tableau algorithm for performing the reasoning and query answering tasks [9].

[9] presents the detailed list of reasoning services for TBox and ABox, along with a more in-depth description of the tableau algorithm.

Over the last decade there has been a broad consensus over the need for rules as well as ontologies. The reason is that ontologies are not strong with regards to the instance level manipulation and reasoning. Logic Programs (LP) [2], on the other hand are strong in reasoning about instances. Moreover, the semantics of the LP is the basis of the important rule systems, such as SQL relational databases, Prolog, and Event-Condition-Action rules. Full logic programs have features that are not expressible in FOL, and thus not in DL [1]. For example consider the procedural attachments, i.e. the association of action-performing procedural invocations with the drawing of conclusions

[2] In this dissertation, I use rule-based reasoning and evaluation of logic programs (LP) interchangeably.

about particular predicates.

In a more detailed discussion, the limitations of DL with respect to LP can be listed as follows [15] [1]:

- A well known problem with the expressivity of DL based representations is the lack of composition operator for roles. For instance, one cannot describe the UncleOf property in terms of the BrotherOf and FatherOf properties.

- DLs lack the support for procedural attachment. Many domain specific reasoners require the use of custom reasoners for IO functions, database calls, etc. this functionality can be easily implemented in logic programs.

- DL cannot represent more than one free variable, while a rule in LP has no such restriction. For instance, consider the rule
  Friend(x,y) $\leftarrow$ Person(x), Person(y).

- DLs do not support n-ary predicates by default, in contrary to LP.

- DLs can not represent classes whose instances are related to an anonymous individual via different properties. For instance, consider the rule
  HomeWorker(?X) $\leftarrow$ Work(?X, ?Y), Live(?X, ?Z), Loc(?Y,?W),Loc(?Z,?W).

- Reasoning in DL considers all logically permissible combination of the facts in the knowledge base and is of high worst case complexity. Rather, through rule based reasoning, one can guide the reasoning engine based on the properties of the domain and deal with the intractability of the reasoning task [30].

In general, it is believed that the benefits of DL can outweigh its shortcomings and extensive efforts have been made in extending knowledge representation power of the DL using rules. In fact, many of the proposals for context aware systems use rules in

conjunction with the semantic context modeling and reasoning. The related work in this regard is presented in the next section.

2.2 Related Work

The Web Ontology Language (OWL) is the main *formal* approach for modeling context [7] [18] [20]. Many projects consider using ontologies for building context aware systems. Ontologies can be used to model the application domain as well as the definition and properties of context. A number of user friendly tools such as $Protégé$ [31] ease the ontology creation and deployment lifecycle. In this section, I will present some of the seminal work in the literature which use ontologies for modeling contextual information. For detailed survey of such systems, please refer to [20][7][18][23][32].

The Context Broker Architecture (CoBrA) is a broker-centric, agent-based architecture for intelligent spaces. CoBrA acquires, maintains and reasons about context, enables knowledge sharing, and detects and resolves inconsistencies in the knowledge [23]. The context knowledge base, specifically, maintains a shared context model for the agents and devices in a smart space. The context data is modelled by a Standard Ontology for Ubiquitous and Pervasive Applications (Soupa) [33]. The design of the Soupa is derived by a number of use cases and consists of two sets of ontology documents, Soupa Core and Soupa Extension. Soupa core describes common concepts such as person, agent, belief-desire-intention (BDI), action, policy, time, space, and event. The Soupa extension defines an extended set of vocabularies for supporting specific pervasive application domains.

Socam [34] uses OWL to describe a hierarchical CONtext ONtology (Conon) [35] which is composed of a high-level part and a domain-specific part. The high level part is an upper ontology for describing generic concepts such as person, location, and activity.

The domain-specific part, on the other hand, describes concepts pertaining to specific domains, e.g. houses, shops, and offices. Such separation of the domains will help in reducing the size of the ontology dealt with, the ease of maintenance, and conceptual comprehension.

The current context in Socam is represented in first-order predicate calculus. For instance, Location(John, school) means that John is currently at school (John and School are already defined to be instances of Person, and Location concepts respectively). This information can be represented in OWL format. Other examples are Temperature (livingRoom, 32) and Activity (Mary, watchTV).

SOCAM allows users to define rules for specific application domains to derive high level context information and consistency checking. The system supports forward chaining, backward chaining, and hybrid execution mode (thanks to the use of Jena rule engine). For instance, consider the following rule:

(?u situation SLEEPING) $\leftarrow$ (?u locatedIn Bedroom) AND (Bedroom lightLevel LOW)

, which decides if a user u is sleeping based on his current location and the lighting level. Furthermore, applications can specify their context aware behaviour through rules such as:

If (John's blood pressure exceeds the threshold) $\vee$
(John's heartbeat is abnormal) $\vee$
(socam:temperature(John, greaterThan(101F)
Then alert hospital emergency department

, which describes a reaction to an emergency situation.

Semantic Space [35] is a similar effort to incorporate semantic web technologies into smart spaces. The authors use separate ontologies for general and domain specific

usages. The context is represented as instances of the ontology (markups). Users can query the desired context from a persistent knowledge store using the RDF Data Query Language (RDQL) [3]. In Semantic Spaces project [35], users can submit the domain specific rules to the system to perform the forward chaining reasoning using the Jena rule engine. This is the only form of reasoning supported and the results of the reasoning are not stored.

Gaia [36] is an infrastructure for intelligent spaces, aiming for integration of physical and virtual entities. It uses concepts from operating systems such as events, signals, and file system, and extends them with context, computing devices, and actuators . Ontologies are used in Gaia to describe various entities involved such as devices, services, and data sources. These ontologies are supposed to be helpful in semantic discovery, matchmaking, interoperability between entities, and human computer interaction [23]. Similar to Socam, Gaia uses first order predicates to represent the contextual axioms. Gaia takes a similar approach to Socam and Smart Spaces, and further employs a priority based mechanism for activating rules, which allows only one rule to fire at a time.

2.3 Discussion

In this chapter I discussed some of the challenging properties of context and drew the conclusion that the formality of the context model has several advantages when considering the properties. Using logic based ontologies bring about a number of advantages when it comes to context modeling. These include facilitation of interoperability, detection of inconsistencies, and a number of reasoning mechanisms which can be used for deriving implicit/abstract information or answering the queries.

Using semantic models has several advantages when it comes to query answering. We can rely on the semantics of the data to not only evaluate the query based on the ex-

[3] www.w3.org/Submission/RDQL/

isting facts, but also by what the existing facts *entail*. In keyword based search, only resources with syntactical similarity will be returned, ignoring the semantic relationships. For example, consider a user querying for *video service* to a system which provides *streaming service*. A purely keyword based query answering mechanism will miss the semantic relationship between the query and the available services [37]. This feature is especially important in pervasive computing environments where the users may not issue the *right* queries due to lack of knowledge, computer skills, or not being focused on the current task.

Despite the mentioned benefits for semantic models, I should note that completely relying on a semantic context model may not be feasible option in practice. In fact, when dealing with low-level raw data, adopting specific non-semantic parsing, filtering, and inference can be more efficient. For instance, consider detecting the activity of the users through several body sensors where ad-hoc methods [2] perform faster and better than first encoding the data into a semantic representation and then relying on the available inference methods.

A new trend in the context-aware community is to use hybrid solutions where ontology based models are used in conjunction with other modeling approaches such as key-value pairs, markup based, or spatial models. The rationale is to reduce the size and the complexity of the ontological knowledge base by employing other models. A representative work is the markup-ontological hybrid model presented in the CARE framework [38]. The authors classify the context data into two types: shallow data and ontology-based. While the former refers to data which is quite static and can be simply represented by key/value pairs. The latter deals with more complicated data such as user activity, and the user environment which may require reasoning to infer more abstract information. The model is based on CC/PP mark-ups which contain references to OWL-DL classes and relations.

In this dissertation, I assume that the semantic representation is used at some stage for context modeling. So the focus is not as much on where we apply the technique or how to blend it with other modeling techniques, but rather on what problems can arise when we rely on it. In the next chapter, we focus on the inefficiency of the query answering with these models when facing changes in the facts.

CHAPTER 3

INCREMENTAL QUERY ANSWERING

As discussed in the previous chapter, a semantic representation brings a number of advantages when modeling context. Thanks to their underlying formalism, it is possible to describe complex context data; share, integrate, and reuse context between sources, and use available reasoning mechanisms for consistency checking or logical inference [7].

Query answering is one of the most important functionalities that empowers the development of context aware systems, applications, and services. In the previous chapter, the ABox instance retrieval query was presented as one of the major reasoning tasks with regards to an OWL knowledge base. If the contextual situations are pre-defined in the ontology, an application would determine if a *situation* holds by retrieving the instances of the class corresponding to the situation. That is, the precise definition of the domain knowledge allows applications to issue queries which determine if a certain

context can be *entailed* in terms of existing definitions. We argue that this type of query is powerful enough for many of the perceivable ubiquitous computing applications.

Here a formal definition of a conjunctive instance retrieval query is presented:

Definition 1. *A conjunctive instance retrieval query is of the form*

$$(answer\ variables) \leftarrow P_1(x_{p1}, y_{p1}) \wedge \cdots \wedge P_m(x_{pm}, y_{pm}) \wedge C_1(x_{c1}) \wedge \cdots \wedge C_n(x_{cn})$$

where P_is and C_is are properties and classes defined in the ontology, respectively. Also, xs and ys are either named individuals or are existentially quantified. In the latter case, they can be one of the answer variables.

We note that for many of the application scenarios, the query results need to be monitored for an extended period of time. This can be due to the nature of the query which can serve a monitoring application. Alternatively, this may be to improve the quality of the results or to decide on their reliability. In fact, as the contextual states are defined in terms of real-world observations, changes are inevitable. In this thesis, we consider this type of queries and refer to them as *subscription* or *long-running* queries.

One issue with OWL based query answering, is the overhead of reasoning under changing instance data. Unfortunately, if standard DL reasoning algorithms and off-the-shelf reasoners are used, scalability issues with respect to query answering are immediately encountered. The current query answering response times using today's tableau-based reasoners are in the order of (tens) of seconds (without considering the consistency checking or query preparation time). This is due to the fact that when a new update is received, the query is re-evaluated over the updated KB [11].

Despite the existence of this problem, proposed context aware systems which leverage on a semantic model barely consider this issue and readily apply the existing tools; while the tools are not optimized for the task at hand. To alleviate the problem, we propose a method which reuses the previous computations and thus alleviates the cost of

reasoning from scratch. This approach is motivated by two observations in ubiquitous computing environments [14]. Firstly, two subsequent contextual states usually do not differ completely, allowing the reuse of the common required computations. Secondly, a considerable portion of envisioned queries are long running subscriptions, making the reuse sensible as long as the queries are valid.

In our method we rely on the Rete [13] algorithm which is used for the evaluation of the production rules in the expert systems. We observe that the algorithm provides a natural support for incremental reasoning, because it saves computation *states* and incrementally incorporates updates to the knowledge base. Furthermore, the algorithm was introduced long ago and there exist a number of highly matured tools such as Jess [39], Jena [40], and Drools [41]. These tools not only implement the original algorithm, but also come with a number of well-known extensions and optimizations which make them suitable in practice.

The method I propose in this dissertation thus aims to leverage on the benefits of the Rete algorithm. To use the algorithm for OWL reasoning and query answering, we need to transform the reasoning tasks in terms of rules. For example, the subsumption expression $C \sqsubseteq D$ would imply that any instance of the class *C* is an instance of class *D*; hence, we can express the semantics as $C(x) \rightarrow D(x)$ for any individual *x*.

Description Logic Program [1] is an expressive subset of the OWL whose semantics is completely representable in terms of Logic Programming rules. In our method, we target this subset of the OWL and assume its usage for modeling context. This allows the translation of the context ontology as well as instance retrieval queries in terms of rules. In this way, reasoning can be performed using the Rete algorithm, allowing incrementally updating query results as changes occur in the observed data.

The rest of this chapter is organized as follows. In section 3.1, I provide more details on Description Logic Programs, its correspondence with OWL. Section 3.2 presents the

Rete algorithm and how we leverage on it for incremental query answering. I further introduce an optimization algorithm in section 3.3. System architecture and experimental studies come in sections 3.4 and 3.5.

3.1 Description Horn Logic Ontologies

A translation between Description Logic (DL) and Logic Programming (LP) establishes a correspondence between two fields of knowledge representation that are largely disparate. This correspondence will allow to transfer results, e.g. reasoning techniques, from one field into the other and nourish the advance of both fields. The Description Logic Programs [1] has been introduced for this purpose. By leveraging DLP, the OWL language family can be supported by a large number of available logic databases and thereby immediately increases the number of systems, which can be utilized for the purposes of Semantic Web applications. In fact, it would be possible to scale beyond toy examples with respect to ABox reasoning problems - a challenging area for OWL reasoning.

The asymmetry of the DLP language primitives, i.e. the fact that most DL class constructors can only be used on the right-hand side of inclusion axioms, makes DLP formally a less expressive ontology language than OWL Lite, which corresponds to $\mathcal{SHIF}(D)$ and theoretically (but not syntactically) allows to use all DLP language constructors on both sides of inclusion axioms and in class equivalence axioms. The authors show [16], however, that most OWL ontologies only use the language primitives provided by DLP. This is mainly due to the fact that most OWL classes are primitive classes and defined through inclusion axioms, i.e. the expressiveness of the right-hand side of inclusion axioms is central in practical usage, while atomic class names are the most frequently used class constructor on the left-hand side of inclusion axioms. Such study

on the expressivity, shows that DLP is useful in practice and should be sufficient for many context aware systems.

Every DLP knowledge base is a syntactically valid OWL knowledge base and is semantically equivalent to a set of *definite Horn clauses* under first-order predicate logic semantics. A clause is said to be definite Horn when it is of form $L_1 \vee \cdots \vee L_k$, where L_i is a literal and exactly one of the literals is positive. A definite Horn clause can be written as a *Horn rule* of the form $H \leftarrow B_1, \wedge \ldots \wedge, B_m$; where H and B_i are atoms and $m \geq 0$. A set of definite Horn rules correspond to a definite Logic Program (LP).

Any OWL knowledge base which can be represented in DLP - i.e. a *Description Horn Logic (DHL)* ontology - can be converted syntactically into an LP. The work in [16] establishes a meaning-preserving translation between DL and LP, along with detailed explanation on how the typical reasoning and inferences available for DLs can be effected in LP. This has been done by relying on the correspondence between DL and First Order Logics (FOL) and the correspondence between FOL and LP.

As the final result, OWL classes are represented as unary predicates and OWL properties as binary predicates. Table 3.1 depicts the recursive mapping of a DHL ontology into its equivalent LP[1]. In this table, *A* represents an atomic class, while C and D are generic classes. The *ObjectIntersectionOf* and *ObjectAllValuesFrom* constructors are mapped to the heads of rules when they appear on the right hand side of an inclusion axiom. The *ObjectUnionOf*, *ObjectSomeValuesFrom*, and *ObjectIntersectionOf* are mapped to the body of the rules when they appear on the left hand side of inclusion axioms.

Example 1. *Consider the following statement from a context ontology schema which defines a Business as an Activity whose all Actors are Employees.*

*SubClassOf (*BusinessMeeting *ObjectIntersectionOf(*Activity

[1]This table also lists which OWL language constructs are covered in the DHL

Table 3.1: The recursive translation of OWL to LP [1].

OWL Functional Syntax [42]	**LP Equivalent**
SubClassOf(C D)	$\mu(D, X) \leftarrow \nu(C, X)$
EquivalentClasses(C D)	$\mu(D, X) \leftarrow \nu(C, X)$ $\mu(C, X) \leftarrow \nu(D, X)$
SubObjectPropertyOf (P Q)	$Q(X, Y) \leftarrow P(X, Y)$
EquivalentObjectProperties (P Q)	$P(X, Y) \leftarrow Q(X, Y)$ $Q(X, Y) \leftarrow P(X, Y)$
InverseObjectProperties (P Q)	$P(Y, X) \leftarrow Q(X, Y)$ $Q(Y, X) \leftarrow P(X, Y)$
TransitiveObjectProperty (P)	$P(X, Z) \leftarrow P(X, Y),$ $P(Y, Z)$
SymmetricObjectProperty (P)	$P(Y, X) \leftarrow P(X, Y)$
ObjectPropertyDomain (P C)	$\mu(C, X) \leftarrow P(X, Y)$
ObjectPropertyRange (P C)	$\mu(C, Y) \leftarrow P(X, Y)$
ClassAssertion (C X)	$\mu(C, X)$
ObjectPropertyAssertion (P X Y)	$P(X, Y)$
Translation Function	**Evaluation**
μ (A,X)	$A(X)$
μ ((ObjectAllValuesFrom (P C)),X)	$\mu(C, Y) \leftarrow R(X, Y)$
μ ((ObjectIntersectionOf (C D)),X)	$\mu(C, X)$, $\mu(D, X)$
ν (A,X)	$A(X)$
ν ((ObjectSomeValuesFrom (P C),X)	$R(X, Y), \nu(C, Y)$
ν ((ObjectIntersectionOf (C D)),X)	$\nu(C, X)$, $\nu(D, X)$
ν ((ObjectUnionOf (C D)),X)	$\nu(C, X)$ or $\nu(D, X)$

*ObjectAllValuesFrom(*hasActor Employee*))*

which is compiled to the following rules:

- *(Rule 1) Activity(X) ← BusinessMeeting(X)*
- *(Rule 2) Employee(Y) ← BusinessMeeting(X) , hasActor(X,Y)*

When translating DL to LP there is the need to transform queries in one domain to the other too. In [16], a complete and formal process for such translation is presented. As for the conjunctive instance retrieval which we consider in this dissertation, the transformation of the queries into their equivalent rules follows a simple translation; where, a new predicate is defined and is placed at the head of the rule. The *arity* of the new predicate is determined by the number of the variables selected by the query.

SPARQL [43] is one of the established query languages for OWL. For illustration purpose, please consider the following conjunctive instance retrieval query expressed in SPARQL, and how it is mapped to a Definite Horn Rule.

Example 2. *Consider the following query which asks for all business meetings starting at a given time:*

```
PREFIX s: <SomeReferenceDomain.com/Context.owl>
SELECT ?X
WHERE{
?X a $c:BusinessMeeting$ .
?X $c:startsAt$ '07/11/2011 09:00 AM' .
     }
```

, which is translated to the rule:

Query1(X) ←

BusinessMeeting(X), startsAt(X, '07/11/2011 09:00 AM')

The translation of DHL ontologies immediately allows users to use the target rule language to extend the ontology with further rules. For instance, the resulting rule base can be reinforced with rules which have procedural attachment; i.e. they can perform IO or network operations in the body or head of the rules (supported by Jess [39]). This not only compensates for the lack of the expressivity resulted by the choice of the DHL sublangage, but also enables the applications with powerful reasoning mechanisms which are not provided even by unrestricted use of OWL. This form of reasoning is also known as *hybrid* and as was mentioned in section 2.2 has been considered in many of the proposals for context aware systems.

We have identified the *Rete* algorithm [13] to be a suitable choice for answering long-running queries when changes happen in the assertional level. Using the translation provided here, it is possible to build a network of computational nodes in the Rete network which allow incrementally maintaining query results as changes occur in the knowledge base. This process will be detailed in the next section.

3.2 Using the Rete Algorithm

Rete [13] is an efficient pattern matching algorithm for implementing production rule systems. The algorithm builds and maintains a network of nodes where each node corresponds to a pattern occurring in the body of a rule. Facts that are added to or removed from the knowledge base are processed by these nodes. If a fact is successfully matched against the conditions represented by one node, it is passed to the nodes directly connected to it.

At the top of the network, the *type* nodes separate individual fact based on their type. Inside the network, we have the *join* or *2-input* nodes which perform finer discriminations and associations between facts. These nodes remember all the facts that arrive in

alpha and *beta* memories, corresponding to their left and right inputs. They produce one output for each ordered pairing of an alpha element and a beta element that passes the tests in that node. When a fact or combination of facts causes all of the patterns for a given rule to be satisfied, a leaf node is reached and the corresponding rule is triggered [44]. In other words, the path from the root to a leaf node defines a complete rule body.

To demonstrate the application of the Rete algorithm, consider figure 3.1, which represents a logical view of the Rete network corresponding to examples 1 and 2 (pages 24 and 26, respectively). The rules corresponding to the ontology schema and the query are used to build the Rete network. In the resulting network, there is a separate type node for each Class and Property used in a rule condition. When an OWL Class or Property assertion (instance data) is received, it is checked by all the type nodes. In this example, all assertions which are not about *BusinessMeeting, startsAt,* and *hasActor* are ignored. Deeper in the network, there exists a join node for each comparison which involves a common variable. The top join node, corresponds to Rule 2, and outputs all instances for which a *BusinessMeeting(X)* as well as *hasActor(X,Y)* exists. The bottom join node represents the rule corresponding to the query and matches all the *BusinessMeeting* which start at *'07/11/2011 09:00 AM'*.

One primary goal of the Rete net is to provide incremental pattern matching. To achieve this, nodes receive notifications about changes. Whenever a new fact is created or deleted, the input node of the appropriate type will release an update *token* on each of its outgoing edges. Such an update token represents changes in the partial matchings stored by the node. Positive update tokens represent newly added facts and negative updates refer to facts being removed from the set. Each node is prepared to receive updates on incoming edges, assess the new situation, determine whether and how the set of stored facts will change, and releases update tokens of its own to signal these changes to its child nodes. This way, the effects of an update will propagate through the

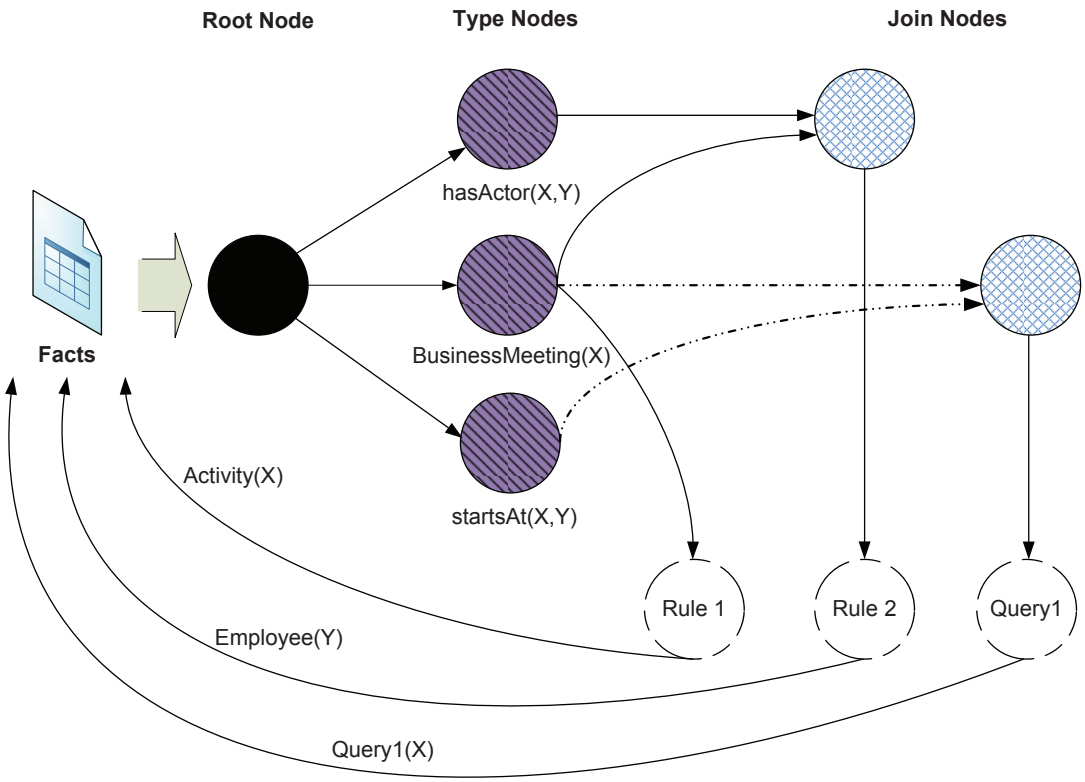


Figure 3.1: The Rete network corresponding to examples 1 and 2

network, eventually influencing the result sets stored in production nodes.

For instance, assume that the initial facts consist of *BusinessMeeting(**meeting1**)*, and later we assert an additional fact *hasActor(**meeting1**, **John Doe**)*. The top join node already contains all the Property assertions about the *BusinessMeeting*, so detecting the match between the two facts can be done without repeating the pattern matching computation done in the initial phase. In this way, the Rete avoids repeating computations over time.

3.3 Adaptive Rule Selection

In this section we provide an optimization for improving the performance of the incremental query answering. The rationale is to prune the rules which are irrelevant to the given query, i.e. their evaluation will not affect the results of the query. As each rule corresponds to a unique path in the Rete network, this will potentially lead to a reduction of execution time.

The decision on whether a rule affects the query results is based on the observation that whether the rule can *affect* the left-hand-side (LHS) of the rule corresponding to the query. We formally define this property as follows:

Definition 2. *Let the rule corresponding to the query Q be of the form $Q \leftarrow B_{q1}, \wedge \ldots \wedge, B_{qm}$. Further assume a rule* R *in the form $H_r \leftarrow B_{r1}, \wedge \ldots \wedge, B_{rm}$. We say that **R** affects **Q** if and only if:*

$H_r \in \{B_{q1}, \ldots, B_{qm}\}$ ***or***

$\exists R\prime \,.\, H_{r\prime} \leftarrow B_{r\prime 1}, \wedge \ldots \wedge, B_{r\prime m}$ *such that*

$H_r \in \{B_{r\prime 1}, \ldots, B_{r\prime m}\}$ ***and*** $R\prime$ affects Q.

Algorithm 1 selects all the rules which satisfy this property. The *schemaRules* corresponds to the rules of the OWL schema. After executing the algorithm, the *selectedRules* will hold the shortlisted rules for maintenance. Given a *Query* in the form of a rule, the algorithm first retrieves all the conditional clauses appearing in its body. For each clause, if it is not checked previously, the algorithm will identify those rules from *schemaRules* which have the clause as one of the consequences in their head. If a rule satisfies this criteria it is recursively checked for other rules which can affect its conditional clauses.

As an example, consider the Rete network presented in Figure 3.1. Starting from the *Query1*, we find that the rule depends on *BusinessMeeting* and *startsAt* predicates. Nevertheless, these two clauses do not appear in the head of any of the existing rules.

Algorithm 1 Adaptive Rule Selection (ARS)

```
input: (Query:Rule, schemaRules:List, selectedRules:List,
checkedClauses:List)
for each bodyElement ∈ Query.body() do
    if bodyElement ∈ checkedClauses then
        continue;
    for each rule ∈ schemaRules do
        if rule ∈ selectedRules then
            continue;
        for each headElement ∈ rule.head() do
            if bodyElement ≡ headElement then
                checkedClauses.add(headElement);
                selectedRules.add(rule);
                call ARS (rule, schemaRules, selectedRules, checkedClauses);
```

As a result, both rules 1 and 2 will be pruned.

The complexity of the algorithm is $O(k \cdot n^2)$, where n is the number of rules and k is the maximum number of elements appearing on the LHS of the rules. n is in the same order of magnitude as the size of the ontology schema (TBox), and k is bound to 2 for the pure usage of rules which are resulted from the schema mapping (i.e. not considering application specific rules). This will render the complexity to be $O(n^2)$. We note that as the rules correspond to mostly static knowledge schema or application specific rules, detecting the dependencies between rules can be performed offline and the dependencies persisted. In this way, the complexity of the algorithm will be linear with respect to the size of the rule base, as each rule is checked only once.

3.4 System Architecture

The required components for the proposed incremental query answering method have been shown in Figure 3.2. At the initialization step the necessary OWL schema (TBox) documents are loaded, where they are translated into their equivalent intensional predicates in the target logic program. After this step, the translation of OWL statements about individuals consists of a simple parsing of the OWL expressions into

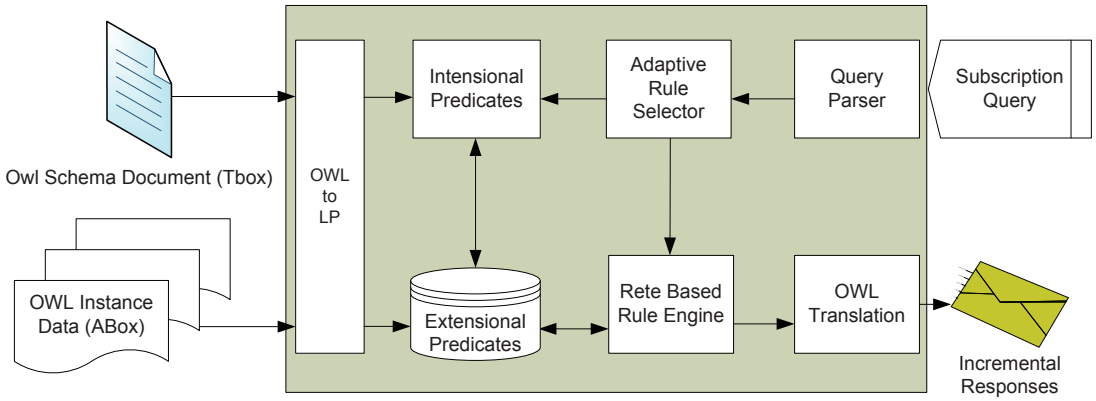


Figure 3.2: Components for Incremental Query Answering

their equivalent extensional predicates of the target LP.

When a query is received, it is first translated into its equivalent rule by the *Query Parser*. It is then fed into the *Adaptive Rule Selector* to prune irrelevant rules depending on their effect on query results. The resulting rules as well as the query are used by the *Rete Based Rule Engine*. The reasoner will fetch necessary information from the extensional knowledge base, does the initial inference and outputs the initial response, along with materialization of the results back into the extensional knowledge base. The results of the query are delivered after a translation by the *OWL Translation* module. As changes occur in the underlying OWL instance base, and thus the extensional predicates, the reasoner will incrementally update the results.

The *extensional predicates*, by default, contains those rules which correspond to the ontology schema. This rule base can be further augmented with the powerful application specific rules (discussed in section 3.1) to compensate for the lack of relative language expressivity. They can further provide features which are not possible when solely relying on OWL based reasoning. Therefore, our method allows *hybrid* ontology and rule-based reasoning for context aware applications.

As can be observed, for our proposal to work, we just need an integration of existing tools and methods and no sophisticated changes in the existing implementations

are required. More precisely, we leverage on the existing implementation of the Rete engines for the execution of the rules. This has been made possible by identifying the relevant fragment of OWL, algorithms, and tools which can work together for delivering the expected results. We highlight this as one of the contributions of this dissertation.

3.5 Experimental Results

In this section I evaluate the performance of the method using the well known Lehigh University Benchmark (LUBM) [45] which is widely used for evaluating the query answering performance of OWL reasoners. To be able to transform the ontology to its equivalent LP representation, we needed to remove the fragments of the ontology schema which make it outside of the DHL fragment. For the transformation of the OWL schema into its equivalent rules we used the OwlTools [46] and its DLPConvert API. The rules corresponding to the ontology are exactly similar to those available at [47].

The 14 queries of the benchmark vary by their input size, their selectivity, their complexity, the hierarchy of data used, and the inference applied. As the queries are quite overlapping, we only select queries 1, 2, and 9 for our experiments. We refer to these three queries as *A, B,* and *C*. In fact, in the OpenRuleBench benchmark [47] these three queries have also been selected and applied for the analysis of various logic programming environments.

Query A asks for all *GraduateStudents* who take a specific course. The query considers just one class and one property and does not assume any hierarchy information or inference. It bears large input and high selectivity. Query B asks for *GraduateStudents* who are employed with the same university they took their undergraduate degree from. It increases in complexity: 3 classes and 3 properties are involved. Additionally, there

Table 3.2: The run time of the rule selection optimization

Query	Selected Rules	Run Time (ms)
A	0 / 98	16
B	13 / 98	20
C	21 / 98	26

exists a triangular pattern of relationships between the objects. Query C asks for all the students whose advisors are the lectures of the courses they are taking. It considers the wide hierarchy of class Faculty and is characterized by the most classes and properties in the query set.

The target of our experiments is to emphasize the effect of a conceptually different incremental query answering method. From a practical point of view, we were interested in two cases: (i) the responsiveness of the method when introducing various amounts of insertions/deletions into the KB and (ii) the effectiveness of the adaptive rule selection optimization.

The LUBM ontology (ABox) consist of a synthetic data set generated by a program and is available from [45]. The experiments measure the performance of our method on two different knowledge bases of 100k and 200k triples (ABox size). For each knowledge base, various amounts of change are introduced in terms of insertions and deletions of facts, and how well the method can keep up with these changes is measured.

For evaluation of the resulting rules we used the Java Expert System Shell (Jess) [39]. The experiments were carried out on a machine with a 32-bit Intel Core2 Duo @2.3GHz processor, 3.2GB of RAM, and Suse Linux 11.3 as OS.

Table 3.2 shows the run time and effectiveness of the algorithm 1 for the three queries. The more rules are excluded from the original ontology rule set, the better is the performance of the query answering in initialization as well as in handling changes. Please note that for the first query, there is no inference involved. In other words, there is no rule which asserts an instance of *GraduateStudent* or *takesCourse* in their head.

As a result, no rules are needed for answering the query, which will greatly alleviate the reasoning cost.

Figures 3.3a and 3.3b, show the initialization time for the three queries over the two knowledge bases.This is the time necessary for performing the initial reasoning and finding the initial query results. As can be observed from the black bars, the application of the Adaptive Rule Selection (presented in 3.3) can highly affect the run time.

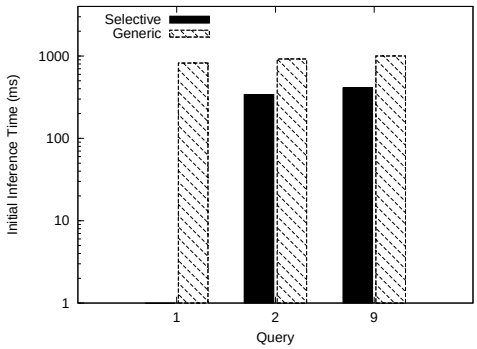
Figures 3.3c, 3.3e, and 3.3g show the query processing time while asserting new facts. Each query is registered with the two knowledge bases and for various amounts of change (10%-50% of the KB size) the runtime is measured. For all the three queries the runtime remains acceptably below one second. It is further reduced by applying the adaptive rule selection optimization. Specifically, for the query A, which is a mere retrieval query, the runtime is zero, as there are no rules involved and the results are materialized in the initialization phase.

Figure 3.3d, 3.3f, and 3.3h show the incremental query processing time while retracting facts from the knowledge bases. As can be observed from the results, the application of the adaptive rule selection optimization greatly improves the performance.

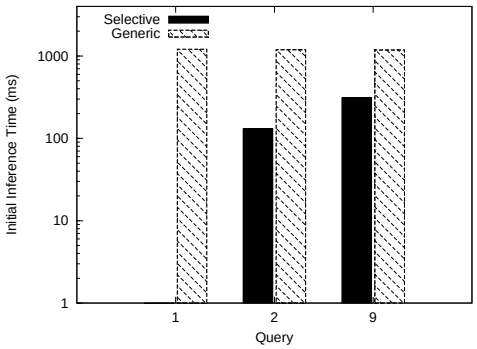
The larger runtime for smaller change rates can be explained by the truth maintenance operations involved. In fact, retracting a fact takes only a little longer than asserting one, on average. But as we have considered the liberal use of logical retraction (for focusing on the worst case scenario), retracting a single fact could result in a kind of *cascade effect*; whereby retracting a single fact would result in many other facts to be removed due to dependencies[2].

There are several considerations when interpreting the applicability of the method with regards to the real-world situations. Firstly, the updates in the form of *new* facts can dominate those corresponding to a *removal* of a previous assertion. For example,

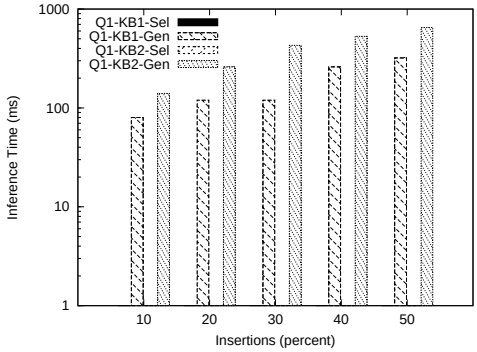
[2]Please refer to our discussions with Jess engineers, available at http://goo.gl/dyDlV



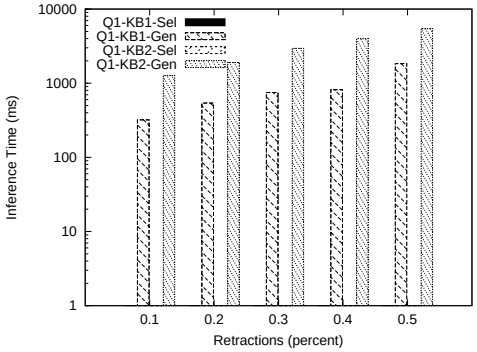
(a) KB1 (Initialization)



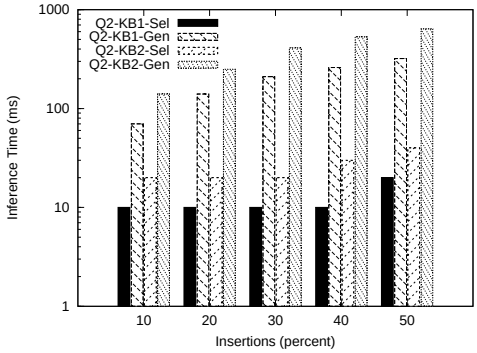
(b) KB2 (Initialization)



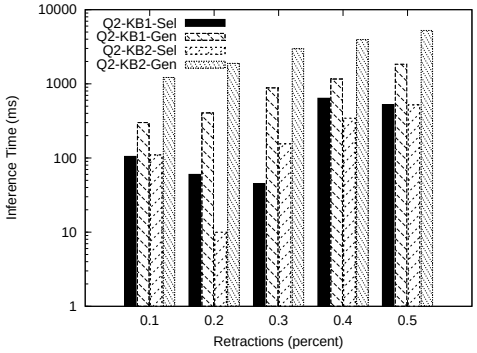
(c) Query A (Additions)



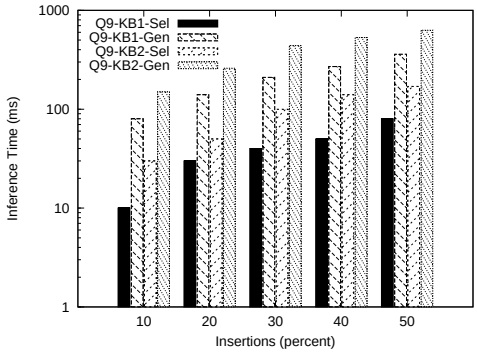
(d) Query A (Retractions)



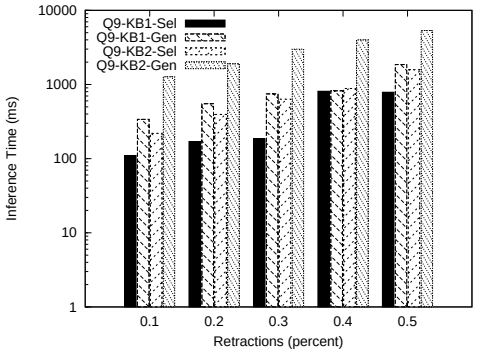
(e) Query B (Additions)



(f) Query B (Retractions)



(g) Query C (Additions)



(h) Query C (Retractions)

Figure 3.3: The query answering time when adding or retracting facts. The 'Selective' case signals the use of the *adaptive rule selection* optimization.

consider the social networking websites such as Twitter[3] where the stream of updates is mostly about new information[4].

Secondly, the update rates considered in our study do not necessarily reflect the actual low-level events of the real world, such as a change in a sensor reading. In fact, an effective solution considers ontology based context modeling and reasoning in conjunction with other modeling and processing approaches, as discussed in section 2.3. If we assume that prior to representing contextual information through well-defined semantics, effective data manipulation and filtering mechanisms are used, we can expect that the update rate in the knowledge base to be much less than the actual changes in real world. This idea is in fact the basis of the work in [48] for combining data stream management systems and semantic reasoners, which will be discussed in the following chapter.

Lastly, to alleviate the issue, we consider a distributed reasoning approach, as will be sketched in chapter 5. The argument is that if the reasoning is performed in a distributed way (e.g. hierarchical), the allegedly high number of delete/update assertions could be potentially reduced within a range which can be handled by our proposed method.

[3] http://www.twitter.com

[4] Twitter itself can be considered as a virtual source of context information.

CHAPTER 4

RELATED WORK

Query answering in ontology based knowledge bases is tightly related to the inference mechanism used. With regards to the dynamic data, the focus has been more on the management and data processing. In fact, research on reasoning over dynamic knowledge is quite limited [49][50][48][11]. In this section we review the work on OWL reasoning/query answering which take the dynamism of the knowledge base into consideration.

The issue of dealing with updates in a knowledge base has been studied in the AI community under belief revision. The main problem is what action should be taken when the update is in contradiction with the existing knowledge base. If the knowledge base is inconsistent, any conclusion can be made from it, which is an undesired state. The simple reaction is to reject the change; in fact, due to lack of a principled approach, this is the choice in the existing ontology management tools [12].

On the other hand, if the inconsistent update is going to be applied, existing beliefs need to be modified. The most influential work in this regard is the widely accepted

AGM model. The work proposes postulates for rational revision and expansion. That is, they specify properties that a contraction or revision operator must have in order to be rational. The main motivation of the AGM is to retain as much information as possible, to have the minimal change [51]. Despite the interest in applying AGM postulates to DL knowledge bases, it has been shown that description logics $\mathcal{SHIF}$ and $\mathcal{SHOIN}$ (corresponding to OWL Lite and OWL DL) do not fall in the AGM-compliant class of logics [11] because the postulates for contraction cannot be satisfied. Moreover, the AGM-based solutions are considered to be complex in practice and not suitable for practical situations [52][53].

In [54], the authors provide a formal analysis of the most basic ABox updates of the form: [¬]a:A and r(a,b). They show that in order to incorporate the new information resulting from an update, the language should support nominals; and further, the @ constructor from hybrid logic or Boolean ABoxes. As a result, both $\mathcal{SHIF}$ and $\mathcal{SHOIN}$ (corresponding to OWL Lite and OWL DL) can not represent the updates without the @ operator. They also highlight that an important issue is the size of the updated ABox, which can be unavoidably exponential both in the size of the original ABox and the new information. Furthermore, the preliminary algorithms provided do not come with neither implementation nor evaluation, and hence we will not discuss them further here.

The work in [11], is motivated by the existence of various dynamic sources of data in the semantic web, including web portals, syndication frameworks (e.g. RSS feeds), and semantic service frameworks (e.g. OWL-S). They provide a syndication framework which matches conjunctive queries submitted by users against a set of dynamic publications which contain rich semantic content. Each publication is essentially a set of ABox assertions which is incorporated in the central knowledge base of the broker.

The main argument is that the central knowledge base should be kept consistent, because otherwise, any information can be derived from it, which is undesirable. For

this purpose they develop an incremental consistency checking algorithm. This is to address the main performance issue in traditional tableau-based OWL reasoners which re-build the entire tableau graph from scratch in the event of an update to the KB. The overall goal is to incrementally update a graph from a previous consistency check.

Having secured the consistency of the central knowledge base of the broker, they consider incrementally resolving registered subscriptions in the syndication framework which are represented as DL conjunctive instance retrieval queries. Rather than considering the entire knowledge base after an update, they develop techniques which aim to reduce the portion of the KB that must be considered as candidate answers. That is to say, the query only needs to be re-evaluated over a subset of the KB over the updated broker's KB.

This work has made some very good steps in addressing the problem of incremental reasoning over DL knowledge bases. In fact, the performance of the proposed incremental reasoning and query answering methods is fairly acceptable, even when facing high change rates. Furthermore, they target a comparatively more expressive fragment of OWL, than the one considered in this dissertation. Nevertheless, this work is limited to pure DL reasoning and the application of rules is not considered. In fact, when solely relying on DL, only those rules can be considered which can be represented back in DL, through a reverse translation method presented in [1]. As we mentioned earlier in section 2.1.1, DLs have certain shortcomings when it comes to reasoning and query answering; while, rules can cover up for the limitation of pure DL based reasoning.

On the other hand, the method presented in this dissertation allows *hybrid* ontology and rule based reasoning. After translation of the ontology schema to the rules, we have *ontology rules*, the evaluation of which corresponds to *ontology-based reasoning*. Now assume that we have additional rules decided by the application logic, which are

not representable in the ontology[1] (i.e. they can be other than Definite Horn rules); we call them *application-specific rules*. For instance, consider the action performing rules (procedural attachment) which allows execution of IO or network operations as the result of rule execution (supported by Jess [39]). In our method, it is possible to add these application specific rules to the ontology rules, enabling ontology as well as rule-based reasoning. In fact, this flexibility in use of rules can compensate for the relative lack of the expressivity, resulted by targeting the DHL fragment of OWL. It should be noted, however, that the arbitrary use of rules may result in the undecidability of the reasoning (and thus query answering), and it would be the responsibility of the knowledge engineers to ensure this property.

In another issue, the method presented in [11], requires fundamental changes in the way DL reasoners are implemented. While the incremental consistency checking method presented is now a part of Pellet [28], the proposed incremental query answering functionality is not available at any of existing publicly-available distributions, at the time of this writing. On the other hand, the method proposed in this dissertation requires no change in existing DL and rule based reasoners and any Rete based rule engine can be utilized.

Therefore, in comparison to [11], our method provides ease of implementation and full support for rules, while targeting a relatively less expressive language.

Another line of research is towards incremental maintenance of materializations of ontological entailments. The classic reasoning tasks are usually triggered when the user queries the system, which is responded by deriving entailed information from asserted information. Under this setting, when the queries are frequent, or the reasoning process is time consuming or complex, the performance may not be acceptable. Materialization amounts to precomputing and storing a set of implicit entailments, so that frequent

[1]The logic programs (LP) and ontology languages have disjoint parts; i.e. LP XOR OWL $\neq \phi$.

and/or crucial queries to the ontology can be answered efficiently [16]. It basically saves the reasoner from recomputing the entailed information for every single query.

In [16], one such approach for incremental maintenance of materialized ontological entailments is presented. It targets the DHL ontologies and follows the same translation method mentioned in section 3.1. After the mapping, the authors leverage on a declarative variant of the Delete and RE-Derive algorithm (DReD) [55] to incrementally maintain the materialization.

The DRed algorithm consists of three steps:

1. Overestimate a deletion: compute all the direct consequences of a deletion

2. Re-derive: neglect those estimated deletions which could be still derived by other facts

3. Insert: insert the new derivations that are consequences of the added facts.

Given an original program, for each predicate P, a maintenance program is derived which consists of seven predicates corresponding to the three steps. The underlying idea is to maintain the materialization of P using the maintenance predicates. The maintenance process starts with a setup, where the maintenance program is created for a given source program and the initial materialization of the intentional predicates is computed. Afterwards, upon a change in the extensional knowledge (ABox), the actual maintenance is triggered.

This approach is fundamentally similar to our approach as it relies the same fragment of OWL and follows the translation of the knowledge base into its equivalent logic program (rules). The difference is on how on what application scenarios are targeted and consequently, how the maintenance is carried out.

The work in [16] mainly aims to speed the response time for *similar queries*; that is, if we process a query and later we receive the same query, there would be minimal

response time. In fact, the method works best for scenarios when consecutive similar queries are issued to the knowledge base. While our approach is also based on materialization and provides the same benefit, we are mainly interested in the *long-running queries*, where we aim to efficiently incorporate changes and update the responses to the registered queries[2].

The method in [16] further leverages on prolog engines for evaluating the queries. The rationale is that we can utilize a wealth of optimizations available in the current logic programming and deductive database systems such as XSB [56]. Nevertheless, we should note the scope of the efficiency of such logic programming systems. They are mainly designed and optimized for answering one-time queries; i.e. queries which are not present over an extended time period. Furthermore, the algorithms are completely agnostic about the nature of the data and whether they change. In fact, the maintenance programs introduced in [16] are to *patch* such tools to enable them to keep track of changes.

Considering the experimental results in [16], we observe that after materialization the cost of accessing the materialized predicates is minimal. Nevertheless, *all* maintenance operations (including those necessary when adding and removing of facts) are more expensive than the cost of evaluating a single query on the original program [16]. As a result, the method is suitable for scenarios where the number of read operations is much higher than the changes in the data. This is one of the arguments made by the authors.

On the other hand, the Rete algorithm is designed to save state by keeping track of all the facts which satisfy nodes within the network. This allows the algorithm to efficiently cater for changes in an incremental fashion when changes occur. While the algorithm is

[2]Long-running or subscribed queries are those which are interested in a continuous assessment of query outcomes. Security, weather and stock monitoring, and social media applications are such examples.

not the best solution for answering one-time queries, we argue that it performs optimally for subscription queries over changing instance data.

LarkC [48] is a distributed framework which tries to address the stream reasoning problem defined as [50]: Given a stream of TBox assertions $T_1, \ldots, T_n$, and/or ABox assertions $A_1, \ldots, A_n$, determine if they entail a certain set of conclusions $C_1, \ldots, C_n$, where each C_i is dependent on the assertions which precede it temporally.

The principal idea is to couple traditional DL reasoners with powerful, reactive, and throughput-efficient Data Stream Management Systems (DSMS). Identifiable periodical changes can be modeled using rules, but the reasoners fall short in handling event driven changes (critical for context aware applications) whose mean time between changes is small. Furthermore, traditional reasoners are good in one-time queries issued explicitly by the user, while in many application scenarios; long-running continuous queries are also required. DSMSs can process transient streams and execute continuous queries; also, they support efficient stream processing, parallel continuous query answering, and adaptive behaviour-which is to decrease the accuracy by introducing higher approximations [48]. The main advantage of this approach is that it uses available techniques for stream processing and DL reasoning.

LarkC consists of five steps which are to be repeated continually until a good enough answer has been calculated. Data is first retrieved, and then abstracted by transforming to logics (e.g. using statistical methods). Relevant data is selected and reasoned about, and the cycle ends by deciding if the answer is appropriate enough. If that is not the case, a new cycle needs to be repeated.

In particular, the abstraction process will use a transcoder which will generate a stream element (α, τ), where α is an RDF statement and the timestamp τ corresponds to the last moment it will be in the observation window. As a result, the system will deal with an RDF stream of the following form:

$$\ldots((subj_i, pred_i, obj_i), \tau_i), ((subj_{i+1}, pred_{i+1}, obj_{i+1}), \tau_{i+1}) \ldots \quad (4.1)$$

The RDF stream is then fed into pre-reasoners which incrementally maintain materialized RDF views (snapshots describing the current state of the system). The views are then given to the reasoners which in turn will calculate a set of answers that remain valid until the pre-reasoner comes up with a new view. The first works on the prereasoner are presented in [52] which is in fact the extension of the incremental view maintenance method mentioned earlier ([16]).

Perhaps the most important contribution of this work is the first steps towards distributed reasoning and coupling of DSMS and logical reasoners. We consider a similar method in our future work which I elaborate more in chapter 5. On the other hand, the proposed incremental reasoning approach is based on [16], which as discussed earlier, is susceptible to the same design unsuitability.

At the end, we mention that there are specific formalisms which incorporate the notion of time, and consider reasoning to be an ongoing process. Representatives are temporal logic, dynamic logic, and active logics. Nevertheless, they have barely been applied in context aware systems and their practicality is not verified. Therefore, we will not include them in this study.

CHAPTER 5

FUTURE WORK

The work presented in this dissertation aimed to provide an easy to use and efficient incremental query answering method for semantic contextual information. The context, as mentioned in chapter 2, has a number of properties which make it challenging for both context modeling and reasoning. We considered one main property of context, the dynamism. In fact, the changes we considered were only in the instance level. Addressing the changes in the knowledge structure (TBox) could be addressed as a future work.

An assumption we made in our work is that all the data necessary for the query answering should be gathered in a central processing node. While being common, this assumption will result in a single processing bottleneck, single point of failure, and suboptimal utilization of network resources (for transferring the data from the context sources). Considerable progress has been made recently in distributed processing, as a means for scaling up logical reasoning on semantic data. The available approaches mostly rely on peer-to-peer architectures [57], Distributed Hash Tables [58], or MapRe-

duce framework [59].

While these efforts establish a good basis for scalable reasoning, they do not consider the distributed computational nodes as sources of information; rather they are nodes to which the original data needs to be transferred and processed locally. This incurs considerable network overhead; and in fact, these approaches are mainly suitable for offline processing of static data, because the calculation times can be very large (even hours).

Here, a distributed reasoning solution with respect to the Coalition is sketched. Coalition [2] is the distributed context aware middleware developed in our group and a schematic representation is depicted in figure 5.1 (end of chapter). It aims to facilitate the development of the context aware applications and services. Coalition has a hierarchical architecture.

Coalition assumes that diverse context data from autonomous physical spaces are categorized and organized into different context domains.

In the bottom, the physical space layer includes all physical entities such as sensors, actuators and computing devices that provide the actual context data. The context data from sources of each *physical space* are funneled through a logical (virtual) device known as Physical Space Gateway (PSG) to the exterior of the physical space. The information coming from PSGs are organized in various *context spaces* such as offices, houses, and clinics, depending on the origin of the context data[1]. Such categorization in the context data management layer aims to provide effective and efficient context data organization, lookup and storage. This layer also provides a query interface to acquire data from the lower layer. The service management layer at the top facilitates the deployment, orchestration, and discovery of the context aware application services which rely on the retrievable contextual information.

[1]The data belonging to the personal handheld devices are categorized in the Person context space.

Now assume that we receive a query whose evaluation depends on a number of distributed context sources, or PSGs in Coalition terminology. To evaluate the query we need to find the information sources which contribute to the information required by the query. In the current implementation of coalition, this task is performed centrally rather being distributed among PSGs; that is, all the required information are retrieved from relevant sources through the respective PSG, are gathered in a central repository, and are checked if they entail the query. Nevertheless, as mentioned earlier, this will result in a suboptimal usage of network resources, and scalability issues.

A better option to evaluate the query is through distribution of query execution, where the necessary information are processed at some intermediary nodes (such as in PSGs), for efficient utilization of network and processing resources as well as reduction in query response time. One promising approach is based on the distributed reasoning method proposed in [60] and [52] where the reasoning process is split up into a network of reasoning *correlators* that are distributed with respect to an optimization goal like minimal network usage or minimal delay. I note that at the time of writing, such extension to Coalition is an on-going process. If we assume that coalition has been reinforced with such distributed query answering mechanism, the method proposed in this dissertation can be used by each of the processing nodes. In this way, each of the nodes will have the ability to incorporate incoming changes incrementally and notify the nodes which rely on its computation results.

We note that, however, distribution the reasoning process can be very challenging in terms of correctness of results and consistency. As a case, each of context sources may have different update rates which can lead to erroneous behavior; for instance, it can lead to concurrent firing of the rules and race conditions [61].

In addition, we did not consider the quality of context in our study. In fact, one major property that distinguishes context information from other type of knowledge in

other domains (e.g. semantic web) is the quality of context. As mentioned in section 2 it is quite important to include freshness, expiry time, validity, and trust into the model. This is to allow uncertain and fuzzy reasoning over the available information.

To represent the quality of context, the axioms in ontology based models can be associated with the relevant metrics, such as confidence or freshness. In fact, there has been considerable effort in incorporating uncertain knowledge into OWL/DL. For instance, Fuzzy Description Logics and Probabilistic Description Logics assign numerical values to the concept/role inclusion axioms or instance/role assertions. For a detailed discussion of the techniques please refer to [15]. Such extensions to OWL and DL enable probabilistic or fuzzy reasoning about contextual data.

Besides considering the distribution and quality of context, there are certain improvements that can be made with respect to the inference method itself, for the sake of robustness. A fundamental assumption in existing reasoning algorithms is that the knowledge base should be consistent, because otherwise any conclusions can be made from it (which is essentially useless). Nevertheless, this is quite restricting in practice where the distribution and heterogeneity of context sources may result in frequent inconsistencies and make it necessary to employ appropriate methods to work around it (rejection or belief revision). The existing work does not consider concurrent access and basically locks the whole knowledge base when handling inconsistencies. This is certainly not a good option in practice where a contextual knowledge base is updated frequently and accessed concurrently. One way to alleviate this deficiency is to consider an *inconsistency tolerant* reasoning mechanism which continues despite some detected inconsistencies within the knowledge base. One approach can be based on justification-based partitioning which enables concurrent belief revision and query answering. This is potentially more robust in handling inconsistency: while certain parts of the KB may be inconsistent and need treatment, others are still accessible.

Additionally, while the adaptive rule selection algorithm presented in section 3.3 provides flexibility to some extent, there are other factors which can be considered to optimize the reasoning process. In fact, the existing reasoners lack the flexibility and adaptability by not considering the *context*. That is, the reasoning is not considered in conjunction with other factors involved such as user, his current task and its importance, quality of service, Service Level Agreement, and the nature of the data dealt with. This information can be used in identifying the usage-driven identification of unnecessary updates, so as to improve the response time of the reasoner.

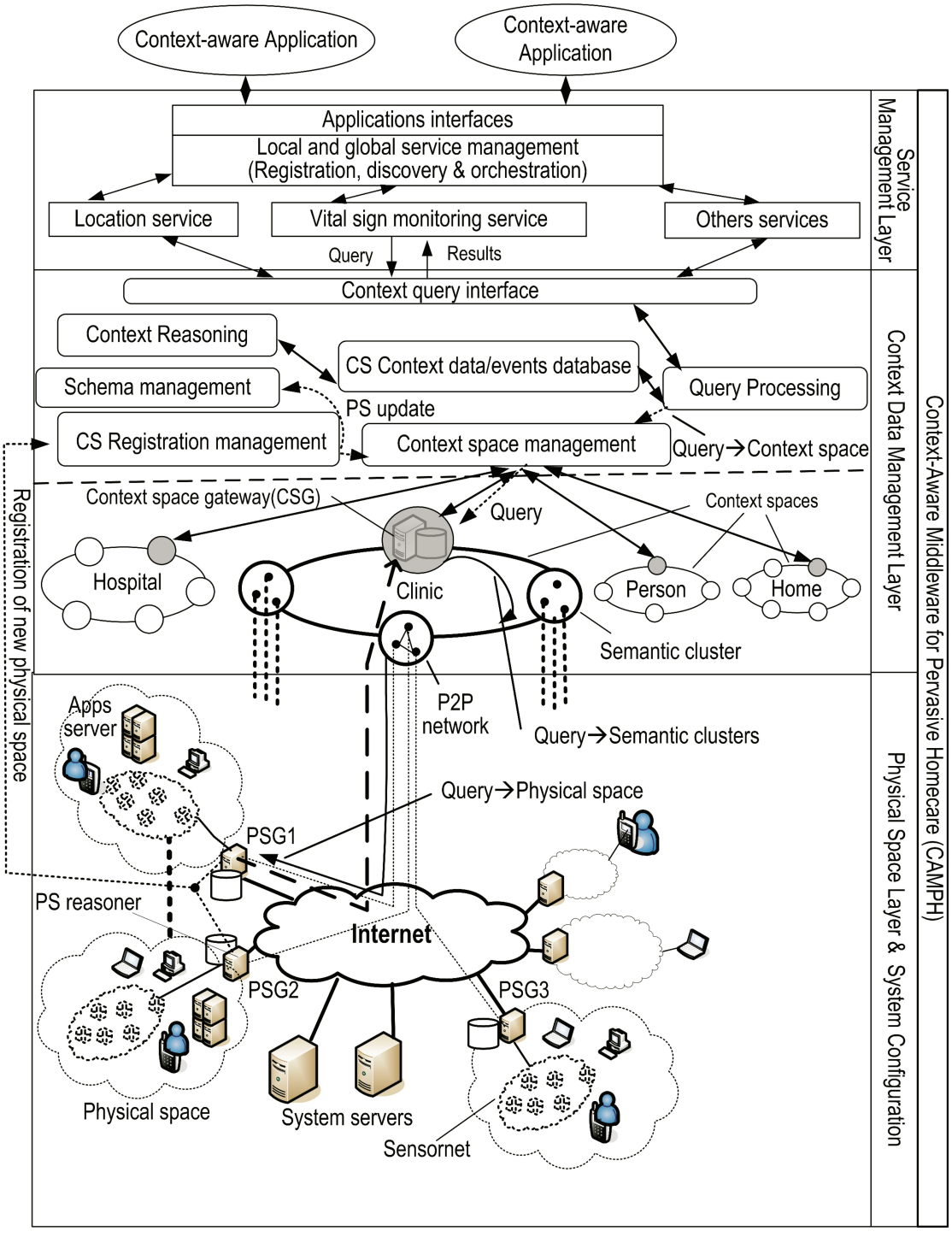


Figure 5.1: The schematic representation of the Coalition context aware middleware [2].

CHAPTER 6

CONCLUSION

Realizing the vision of Ubiquitous computing requires seamless integration of software systems with our daily lives. In other words, they need to be context aware. In a real-world setting, contextual information can be retrieved from distributed heterogeneous sources. This can make application development a challenging task, due to a number of issues including interoperability, scalability, and quality of service. The Ubiquitous computing community increasingly advocates taking a formal approach at some stage in context modeling, so as to alleviate the interoperability issue. The Web Ontology Language (OWL) is the result of a joint effort between academia and industry, making it the standard notion for representing semantic information in the Web. The well studied syntax and semantics of the language has motivated many research projects to take advantage of it for modeling context information. Using OWL specifically enables a number of automated reasoning services which can be used for query answering and deducing new information. Nevertheless, most efforts have readily used the language without tailoring it for the specific needs of context aware systems.

In fact, my research process started with the study of ubiquitous and pervasive computing, context aware systems, and the huge body of research on context modeling and reasoning. I then observed that a common consensus in the community is leveraging a formal model, mostly based on a shared understanding of the domain: an ontology. Having this assumption, I asked the question of "what problems can arise in practice, if we use a formal semantic representation of context."

One of the major issues with the use of OWL is due to the static nature of the existing reasoners. In practice, context information is retrieved from sources which may produce data quite frequently. Traditionally, most of the well-established reasoning (and thus query answering) methods are designed with static or slowly-changing facts in the knowledge base. In this way, upon a change in the observed facts, the query needs to be re-evaluated from the beginning to reflect the new changes. In this dissertation I focused on this problem and proposed an incremental query answering solution which aims to alleviate the cost of reasoning from scratch.

To take advantage of the proposed technique, a well-known translation from an OWL ontology schema to its equivalent Horn rules is required. After the mapping, any Rete based rule reasoner can be used; giving the flexibility in reusing existing tools. The technique can be applied to the fragments of the OWL which can be expressed in terms of rules, namely Description Horn Logics (DHL). In fact, many of the existing ontologies barely use the OWL language constructs which are outside the fragment [16]. The translation of DHL ontologies immediately allows users to extend the ontology with further application specific rules. Such augmentation provides an immediate remedy for the reduction in the expressiveness; if the added rules do not contain intractable language features. Besides, we provided a selection optimization which has linear time complexity in the size of the resulting rule base and can improve the query answering performance significantly. Through the experiments with a well-known benchmark we

showed our method produces acceptable results and thus we argue that our method has addressed the problem defined earlier.

We believe that the method proposed in this dissertation has a great potential for the development of context aware systems; in that, it requires minimum changes and the researches and practitioners can focus on other problems arising in practice. While we did not consider the distribution and quality of context into our study, we argue that these two aspects have a subjective nature, where the suitability of the solution highly depends on the used infrastructure and specific considerations of application domain. Taking distribution as an example, the optimal distributed reasoning method for Coalition [2] would be different from that for Solar [62] which takes a different model for intermediary data aggregation. The same argument applies for the quality of context, e.g. whether we want to have a probabilistic or fuzzy reasoning and whether we want to include the freshness and trust aspects of the context data in the reasoning process. The dynamism of the context data, however, has an objective nature, which can be consistently assumed similar for application domains, albeit with different change rates.

As for the next step of this research, it will be thus very interesting - and necessary - to consider all the context properties for semantic query answering in a concrete domain, e.g. in the context of the Coalition middleware [2].

BIBLIOGRAPHY

[1] B. N. Grosof, I. Horrocks, R. Volz, and S. Decker, "Description logic programs: combining logic programs with description logic," in *Proceedings of the 12th international conference on World Wide Web*. Budapest, Hungary: ACM, 2003, pp. 48–57–.

[2] H. Pung, T. Gu, W. Xue, P. Palmes, J. Zhu, W. Ng, C. Tang, and N. Chung, "Context-aware middleware for pervasive elderly homecare," *IEEE Journal on Selected Areas in Communications*, vol. 27, no. 4, pp. 510–524, 2009.

[3] M. Weiser, "The computer for the 21st century," in *Human-computer interaction*, R. M. Baecker, J. Grudin, W. A. S. Buxton, and S. Greenberg, Eds. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995, ch. The computer for the 21st century, pp. 933–940.

[4] H. Lei, "Context awareness: a practitioner?s perspective," *Ubiquitous Data Management, International Workshop on*, vol. 0, pp. 43–52, 2005.

[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, pp. 19–26, December 2007.

[6] M. Satyanarayanan, "Pervasive computing: vision and challenges," *Personal Communications, IEEE*, vol. 8, no. 4, pp. 10 –17, Aug. 2001.

[7] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, and D. Riboni, "A survey of context modelling and reasoning techniques," *Pervasive and Mobile Computing*, vol. 6, no. 2, pp. 161 – 180, 2010, context Modelling, Reasoning and Management.

[8] (2003) Owl web ontology language overview. [Online]. Available: http://www.w3.org/TR/owl-features/

[9] F. Baader, *The description logic handbook : theory, implementation, and applications*. Cambridge: Cambridge University Press, 2010.

[10] (2004) Resource description framework (rdf): Concepts and abstract syntax. [Online]. Available: http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/

[11] F. C. Halaschek-Wiener, "Expressive syndication on the web using a description logic-based approach," Ph.D. dissertation, University of Maryland, College Park, College Park, MD, USA, 2007, aAI3297367.

[12] G. De Giacomo, M. Lenzerini, A. Poggi, and R. Rosati, "On the update of description logic ontologies at the instance level," in *proceedings of the 21st national conference on Artificial intelligence - Volume 2*. AAAI Press, 2006, pp. 1271–1276.

[13] C. L. Forgy, "Rete: A fast algorithm for the many pattern/many object pattern match problem," *Artificial Intelligence*, vol. 19, no. 1, pp. 17 – 37, 1982.

[14] M. Oliya and H. K. Pung, "Towards incremental reasoning for context aware systems," in *Advances in Computing and Communications*, ser. Communications in Computer and Information Science. Springer Berlin Heidelberg, 2011, vol. 190, pp. 232–241.

[15] J. Zhao, *Advances in Semantic Computing*. Technomathematics Research Foundation (TMRF), 2010, vol. 2, ch. 1, pp. 1–22.

[16] R. Volz, S. Staab, and B. Motik, "Incrementally maintaining materializations of ontologies stored in logic databases," in *Lecture Notes in Computer Science*, S. Spaccapietra, E. Bertino, S. Jajodia, R. King, D. McLeod, M. E. Orlowska, and L. Strous, Eds. Springer Berlin / Heidelberg, 2005, vol. 3360, pp. 1–34–.

[17] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, "Towards a better understanding of context and context-awareness," in *Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, ser. HUC '99. London, UK: Springer-Verlag, 1999, pp. 304–307.

[18] A. Bikakis, T. Patkos, G. Antoniou, and D. Plexousakis, "A survey of semantics-based approaches for context reasoning in ambient intelligence," in *Constructing Ambient Intelligence*, ser. Communications in Computer and Information Science, M. Muhlhauser, A. Ferscha, and E. Aitenbichler, Eds. Springer Berlin Heidelberg, 2008, vol. 11, pp. 14–23.

[19] O. Lassila and D. Khushraj, "Contextualizing applications via semantic middleware," in *Proceedings of the The Second Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 183–191.

[20] M. Perttunen, J. Riekki, and O. Lassila, "Context representation and reasoning in pervasive computing: a review," *International Journal of Multimedia and Ubiquitous Engineering*, vol. 4, no. 4, pp. –, 2009.

[21] K. Henricksen and J. Indulska, "Modelling and using imperfect context information," in *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, ser. PERCOMW '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 33–.

[22] T. Strang and C. Linnhoff-Popien, "A context modeling survey," in *1st Int. Workshop on Advanced Context Modelling, Reasoning and Management*, 2004, pp. –.

[23] J. Ye, L. Coyle, S. Dobson, and P. Nixon, "Ontology-based models in pervasive computing systems," *Knowl. Eng. Rev.*, vol. 22, pp. 315–347, December 2007.

[24] (2010) Rdf - semantic web standards. [Online]. Available: http://www.w3.org/RDF/

[25] J. Heflin. (2004) Owl web ontology language use cases and requirements. [Online]. Available: http://www.w3.org/TR/webont-req/

[26] M. Luther, Y. Fukazawa, M. Wagner, and S. Kurakake, "Situational reasoning for task-oriented mobile service recommendation," *Knowl. Eng. Rev.*, vol. 23, pp. 7–19, March 2008.

[27] S. Boehm, J. Koolwaaij, M. Luther, B. Souville, M. Wagner, and M. Wibbels, "Introducing iyouit," in *The Semantic Web - ISWC 2008*, ser. Lecture Notes in Computer Science, A. Sheth, S. Staab, M. Dean, M. Paolucci, D. Maynard, T. Finin, and K. Thirunarayan, Eds. Springer Berlin / Heidelberg, 2008, vol. 5318, pp. 804–817.

[28] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical owl-dl reasoner," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 5, no. 2, pp. 51 – 53, 2007, software Engineering and the Semantic Web.

[29] V. Haarslev and R. Muller, "Racer system description," in *Proceedings of the First International Joint Conference on Automated Reasoning*, ser. IJCAR '01. London, UK, UK: Springer-Verlag, 2001, pp. 701–706.

[30] R. Brachman, *Knowledge representation and reasoning*. Amsterdam ;;Boston: Morgan Kaufmann, 2004.

[31] (2011) The protege ontology editor and knowledge acquisition system. [Online]. Available: http://protege.stanford.edu/

[32] T. Eiter, G. Ianni, A. Polleres, R. Schindlauer, and H. Tompits, "Reasoning with rules and ontologies," in *Lecture Notes in Computer Science*, P. Barahona, F. Bry, E. Franconi, N. Henze, and U. Sattler, Eds. Springer Berlin / Heidelberg, 2006, vol. 4126, pp. 93–127–.

[33] H. Chen, F. Perich, T. Finin, and A. Joshi, "Soupa: Standard ontology for ubiquitous and pervasive applications," *Mobile and Ubiquitous Systems, Annual International Conference on*, vol. 0, pp. 258–267, 2004.

[34] T. Gu, H. K. Pung, and D. Q. Zhang, "Toward an osgi-based infrastructure for context-aware applications," *IEEE Pervasive Computing*, vol. 3, pp. 66–74, October 2004.

[35] X. Wang, J. S. Dong, C. Chin, S. Hettiarachchi, and D. Zhang, "Semantic space: An infrastructure for smart spaces," *IEEE Pervasive Computing*, vol. 3, pp. 32–39, July 2004.

[36] M. Roman, C. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt, "A middleware infrastructure for active spaces," *IEEE Pervasive Computing*, vol. 1, pp. 74–83, October 2002.

[37] T. Chaari, D. Ejigu, F. Laforest, and V.-M. Scuturici, "A comprehensive approach to model and use context for adapting applications in pervasive environments," *Journal of Systems and Software*, vol. 80, no. 12, pp. 1973 – 1992, 2007, selected papers from the International Conference on Pervasive Services (ICPS 2006).

[38] A. Agostini, C. Bettini, and D. Riboni, "A performance evaluation of ontology-based context reasoning," in *Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications Workshops*, ser. PERCOMW '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 3–8.

[39] Jess, the rule engine for the java platform. [Online]. Available: www.jessrules.com/

[40] (2011) Jena a semantic web framework for java. [Online]. Available: http://jena.sourceforge.net/

[41] (2011) Drools - the business logic integration platform. [Online]. Available: http://www.jboss.org/drools

[42] (2009) Owl 2 web ontology language; structural specification and functional-style syntax. [Online]. Available: http://www.w3.org/TR/owl2-syntax/

[43] (2008) Sparql query language for rdf. [Online]. Available: http://www.w3.org/TR/rdf-sparql-query/

[44] E. Friedman-Hill, *Jess in Action: Java Rule-Based System*. Manning Publications, 2003.

[45] The lehigh university benchmark (lubm). [Online]. Available: http://swat.cse.lehigh.edu/projects/lubm/

[46] (2006) Kaon2 owl tools. [Online]. Available: http://km.aifb.kit.edu/projects/owltools/

[47] S. Liang, P. Fodor, H. Wan, and M. Kifer, "Openrulebench: an analysis of the performance of rule engines," in *Proceedings of the 18th international conference on World wide web*, ser. WWW '09. New York, NY, USA: ACM, 2009, pp. 601–610.

[48] E. Della Valle, S. Ceri, D. Barbieri, D. Braga, and A. Campi, "A first step towards stream reasoning," in *Future Internet FIS 2008*, ser. Lecture Notes in Computer Science, J. Domingue, D. Fensel, and P. Traverso, Eds. Springer Berlin / Heidelberg, 2009, vol. 5468, pp. 72–81.

[49] T. Weithoner, T. Liebig, M. Luther, S. Bohm, F. von Henke, and O. Noppens, "Real-world reasoning with owl," in *The Semantic Web: Research and Applications*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2007, vol. 4519, pp. 296–310.

[50] G. Unel and D. Roman, "Stream reasoning: A survey and further research directions," in *Lecture Notes in Computer Science*, T. Andreasen, R. Yager, H. Bulskov, H. Christiansen, and H. Larsen, Eds. Springer Berlin / Heidelberg, 2009, vol. 5822, pp. 653–662–.

[51] P. Gardenfors, *Belief revision*. Cambridge: Cambridge university press, 2003.

[52] D. Barbieri, D. Braga, S. Ceri, E. Della Valle, and M. Grossniklaus, "Incremental reasoning on streams and rich background knowledge," in *The Semantic Web: Research and Applications*, ser. Lecture Notes in Computer Science, L. Aroyo,

G. Antoniou, E. Hyvanen, A. ten Teije, H. Stuckenschmidt, L. Cabral, and T. Tudorache, Eds. Springer Berlin / Heidelberg, 2010, vol. 6088, pp. 1–15.

[53] H. Stuckenschmidt, S. Ceri, E. D. Valle, and F. van Harmelen, "Towards expressive stream reasoning," in *Semantic Challenges in Sensor Networks*, ser. Dagstuhl Seminar Proceedings, K. Aberer, A. Gal, M. Hauswirth, K.-U. Sattler, and A. P. Sheth, Eds., no. 10042. Dagstuhl, Germany: Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2010.

[54] H. Liu, C. Lutz, M. Milicic, and F. Wolter, "Updating description logic aboxes," in *International Conference of Principles of Knowledge Representation and Reasoning(*. AAAI, 2006, pp. –.

[55] M. Staudt and M. Jarke, "Incremental maintenance of externally materialized views," in *Proceedings of the 22th International Conference on Very Large Data Bases*, ser. VLDB '96. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996, pp. 75–86.

[56] (2011) Xsb logic programming and deductive database system. [Online]. Available: http://xsb.sourceforge.net/

[57] G. Tao, P. Hung Keng, and Z. Daqing, "Peer-to-peer context reasoning in pervasive computing environments," in *Pervasive Computing and Communications, 2008. PerCom 2008. Sixth Annual IEEE International Conference on*, 2008, pp. 406–411–.

[58] S. Kotoulas, E. Oren, and F. v. Harmelen, "Mind the data skew: distributed inferencing by speeddating in elastic regions," in *Proceedings of the 19th international conference on World wide web*. Raleigh, North Carolina, USA: ACM, 2010, pp. 531–540–.

[59] J. Urbani, S. Kotoulas, J. Maassen, F. van Harmelen, and H. Bal, "Owl reasoning with webpie: Calculating the closure of 100 billion triples," in *Lecture Notes in Computer Science*, L. Aroyo, G. Antoniou, E. Hyvnen, A. ten Teije, H. Stuckenschmidt, L. Cabral, and T. Tudorache, Eds. Springer Berlin / Heidelberg, 2010, vol. 6088, pp. 213–227–.

[60] S. Rizou, K. Hussermann, F. Drr, N. Cipriani, and K. Rothermel, "A system for distributed context reasoning," in *Proceedings of the 2010 Sixth International Conference on Autonomic and Autonomous Systems*, ser. ICAS '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 84–89.

[61] M. Sama, S. Elbaum, F. Raimondi, D. S. Rosenblum, and Z. Wang, "Context-aware adaptive applications: Fault patterns and their automated identification," *IEEE Transactions on Software Engineering*, vol. 36, pp. 644–661, 2010.

[62] G. Chen, M. Li, and D. Kotz, "Data-centric middleware for context-aware pervasive computing," vol. 4. Amsterdam, The Netherlands, The Netherlands: Elsevier Science Publishers B. V., April 2008, pp. 216–253.

Appendix A - Publications

Materials in this thesis are mainly revised from the following list of papers:

1. **Mohammad Oliya**, Jian Zhu, Hung Keng Pung, and Antoine Veillard, "Incremental Query Answering over Dynamic Contextual Information", *In 23rd IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, 2011 .

2. **Mohammad Oliya**, and Hung Keng Pung, "Towards incremental reasoning for context aware systems", *in Advances in Computing and Communications,* ser. Communications in Computer and Information Science. Springer Berlin Heidelberg, 2011, vol. 190, pp. 232241.

In addition, during my M.Sc. study, I have collaborated to other research on context aware systems, application, and services:

1. Jian Zhu, **Mohammad Oliya**, Hung Keng Pung and Wai Choong Wong, "SOLE: Context-Aware Sharing of Living Experiences", *In International Journal of Pervasive Computing and Communications (IJPCC)*, Volume 7, Issue 1, 2011.

2. Jian Zhu, Penghe Chen, Hung Keng Pung,**Mohammad Oliya**, Shubhabrata Sen and Wai Choong Wong, "Coalition: A Platform for Context-Aware Mobile Application Development", *In Ubiquitous Computing and Communication Journal (UBICC)*, Volume 6, Issue 1, 2011.

3. Jian Zhu, **Mohammad Oliya** and Hung Keng Pung, "Service Discovery for Mobile Computing: Classifications, Considerations and Challenges". *in Handbook of Mobile Systems Applications and Services (Editors: Anup Kumar and Bin Xie)*, CRC Press, Taylor and Francis Group, USA, 2011.

4. Jian Zhu, Hung Keng Pung, **Mohammad Oliya** and Wai Choong Wong, "A Context Realization Framework for Ubiquitous Applications with Runtime Support", *in IEEE Communications Magazine*, 2011.

5. Jian Zhu, **Mohammad Oliya**, Hung Keng Pung and Wai Choong Wong, "LASPD: A Framework for Location-Aware Service Provision and Discovery in Mobile Environments", *In Proceedings of IEEE 6th Asia Pacific Services Computing Conference (APSCC)*, pp. 218–225, Hangzhou, China, 2010.

6. Jian Zhu, **Mohammad Oliya**, Hung Keng Pung and Wai Choong Wong, "SOLE: Context-Aware Sharing of Living Experience in Mobile Environments", *In Proceedings of 8th International Conference on Advances in Mobile Computing and Multimedia (MoMM)*, pp. 366–369, Paris, France, 2010.

7. Jian Zhu, **Mohammad Oliya** and Hung Keng Pung, "A Pragmatic Approach to Location-Aware Service Organization and Discovery", *In Proceedings of IEEE 28th International Performance, Computing, and Communications Conference (IPCCC)*, pp. 272–279, Arizona, USA, 2009.