Incremental Query Answering over Dynamic Contextual Information

Mohammad Oliya, Jian Zhu, Hung Keng Pung
School of Computing
National University of Singapore
Singapore
Email: {oliya, zhujian, punghk}@comp.nus.edu.sg

Abstract—Context awareness is one of the key requirements for realizing the vision of ubiquitous computing. Formal representation of context information fosters interoperability, eases development of context-aware applications, and enables a number of reasoning mechanisms. The proposed formal models are mainly based on the Web Ontology Language (OWL). Nevertheless, traditional OWL reasoning and query answering methods are not optimized for fast changes in context data. In this paper, we present an incremental query answering method by leveraging on the Rete algorithm. The context ontology as well as the queries are translated into rules which can be further augmented with application specific rules. These rules are used to build the Rete network which incrementally maintains the query results as changes occur in the context data. We further employ a heuristic to adaptively prune the rules which do not affect the query results. Empirical results suggest the practicality of our incremental query answering approach.

Keywords—incremental query answering; context awareness; ubiquitous computing; ontology; rules; Rete;

I. INTRODUCTION

In the vision of ubiquitous computing, the world is saturated with pervasive computing and communication technologies. To be minimally intrusive, a ubiquitous computing environment must be context-aware. Useful context information can be retrieved from hardware sensing devices (e.g. location, temperature, ECG) or information providers (e.g. road traffic data, user’s calendar or flight information). They can be used for tailoring the set of application-relevant data, discovering services, and adapting user interfaces [1].

The ubiquitous computing community increasingly understands benefits of formal context information modeling. Firstly, it alleviates the heterogeneity of context sources, which in turn fosters reuse and sharing of context information which can be expensive to gather, evaluate and maintain. Besides, a formal context model reduces the complexity of context-aware applications and simplifies their development and maintenance. A well-defined semantics also enables a number of reasoning services are also enabled which can be used to derive implicit or abstract information, answer queries, and detect the inconsistencies peculiar to context data [2].

The dominant formal approach for modeling context is by using the Web Ontology Language (OWL) 1. This language is characterized by the formal semantics of Description Logics (DL) [3] and its RDF-based serializations 2. However, the main limitation in using OWL as the underlying representation model is in the overhead of DL reasoning under changing data. Traditionally, reasoning on an updated Knowledge Base (KB) 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 [4]. This is impractical for Ubicomp environments with fast changes in context data and real-time query answering requirements. In this paper we propose an incremental query answering method for OWL based contextual information.

In our approach we leverage on the Rete [5] algorithm which is widely used for the evaluation of production rules in expert systems. For this purpose, the ontology schema as well as the queries are transformed into their equivalent definite Horn rules, i.e. rules with only one literal at the head. Later they are used in building the Rete network. In this way, the previous computations are reused and the query results are maintained incrementally as changes occur in the underlying instance data. As evaluation of the resulting rules can be exhaustive, we further introduce a heuristic to prune the set of rules which do not affect query results.

As we rely on the Rete algorithm, our approach promises the complimentary incorporation of application-specific rules, which are one of the main inference methods in context aware systems. Moreover, the incremental query answering functionality can be achieved without altering the existing implementations. We believe that our approach provides a fare trade off between expressivity of the underlying language, support for application specific rules, and the ease of implementation.

II. DESCRIPTION HORN LOGIC ONTOLOGIES

The Web Ontology Language (OWL) is the main formal approach for modeling context [1][2]. In principle, an OWL knowledge base consists of two parts. The schema (TBox) is the intentional knowledge and defines knowledge structure

1http://www.w3.org/TR/owl-features/
2http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/
in terms of concepts and their relationships. The instance data (ABox) is the extensional knowledge and describes the actual individuals and relationships.

Description Logic Programs (DLP) [6] aim to integrate rules with the ontology layer by compiling ontology definitions into a logic program which can later be extended with application specific rules. This is specifically important, as rule based reasoning is one of the main inference methods in context aware systems ([1][2]). Besides, from a querying perspective, ontology languages are quite weak with regards to instances while rules offer extensive facilities for instance reasoning [6].

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 \( L_1 \lor \cdots \lor L_k \), 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 \( H \leftarrow B_1, \land \cdots \land B_m \); where \( H \) and \( B_i \) are atoms and \( m \geq 0 \). A set of definite Horn rules correspond to a definite Logic Program (LP).

An OWL knowledge base is Description Horn Logic (DHL), if and only if it can be represented in DLP. The transformation is a recursive process and the interested reader may refer to [6] for more elaboration.

III. INCREMENTAL QUERY ANSWERING

In our method, we first translate the ontology as well as the query into their equivalent definite Horn rules. We target the DHL fragment of the Web Ontology Language and follow the translation process presented in [6]. We further consider conjunctive instance retrieval SPARQL\(^3\) queries which determine all the individuals in the knowledge base that are instances of a given class, property, or both. Transforming these queries into their equivalent rules follows a simple translation. In this process, a new predicate is defined and is placed at the head of the rule. The arity of the predicate is determined by the number of the variables selected by the query.

The Rete algorithm [5] 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. Each node has one or two inputs and any number of outputs. 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. In this way, the path from a type node to a leaf node defines a complete rule body. The results of a query are the instances of the predicate appearing at the head of the query. Besides, as the Rete algorithm poses no restriction on the rules to use, one can augment the set of rules with application specific rules.

One primary goal of the Rete algorithm is to provide incremental pattern matching. To achieve this, input 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 network, eventually influencing the result sets stored in production nodes.

The Rete algorithm performs inference in a forward chaining manner, which despite employed optimizations, can be costly in terms of computation and memory usage. To alleviate the issue, we develop a heuristic which prunes the rules which are irrelevant to the given query, i.e. their evaluation will not affect the query results. As each rule corresponds to a unique path in the Rete network, this will potentially lead to a reduction of memory usage and execution time. The procedure is detailed in Algorithm III.

The \texttt{schemaRules} corresponds to the rules of the OWL schema. After executing the algorithm, the \texttt{selectedRules} will hold the shortlisted rules for maintenance. Given a \texttt{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 \texttt{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.

\begin{algorithm}[H]
\caption{Adaptive Rule Selection (ARS)}
\begin{algorithmic}
\State \texttt{input: (Query:Rule, schemaRules:List, selectedRules:List, checkedClauses:List)}
\For{each \texttt{bodyElement} \in Query.body()}
\If{\texttt{bodyElement} \in checkedClauses}
\State continue;
\Else
\EndIf
\EndFor
\For{each \texttt{rule} \in schemaRules}
\If{\texttt{rule} \in selectedRules}
\State continue;
\Else
\For{each \texttt{headElement} \in rule.head()}
\If{\texttt{bodyElement} \equiv \texttt{headElement}}
\State checkedClauses.add(\texttt{headElement});
\State selectedRules.add(rule);
\State call ARS (\texttt{rule}, schemaRules, selectedRules, checkedClauses);
\EndIf
\EndFor
\EndIf
\EndFor
\end{algorithmic}
\end{algorithm}

\(^3\)http://www.w3.org/TR/rdf-sparql-query/
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 and 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.

IV. EXPERIMENTAL RESULTS

The target of the 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 heuristic. We leverage on the well known Lehigh University Benchmark (LUBM)\(^4\) which is widely used for evaluating the query answering performance of OWL reasoners. For the transformation of the OWL into its equivalent logic program we used the OwlTools\(^5\) and its DLPConvert API. For evaluation of the resulting rules we used the Java Expert System Shell (Jess)\(^6\).

The experiments measure the performance of our method on two different knowledge bases of 100k and 200k triples. For each knowledge base, we introduce various amounts of change in terms of insertions and deletions, and measure how well the method can keep up with these changes. Due to space constraint, we only consider query number 9 which asks for all the students whose advisors are the lecturers 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 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.

Figure 1a shows 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 performance is evaluated. For all the three queries the runtime remains acceptably below one second. It is further reduced by applying the adaptive rule selection heuristic. Figure 1b shows the incremental query processing time while retracting facts from the knowledge bases. The larger runtime for smaller change rates can be explained by the truth maintenance operations involved. In fact retracting a single fact could result in a ripple effect which leads to firing of a chain of depending rules. As can be observed from the results, the application of the adaptive rule selection heuristic greatly improves the performance, because unnecessary rules are not involved.

V. RELATED WORK

The research on reasoning over dynamic knowledge is quite limited [7][8][4]. The efforts can be categorized into two groups. The first tries to achieve incremental reasoning and query answering by altering the tableau decision process for detecting inconsistencies in a DL/OWL knowledge base [3]. The seminal work in this strain is [4], providing a Web syndication framework which matches conjunctive instance retrieval queries against a stream of rich semantic content. This work, however, is limited to pure DL reasoning and the application of rules is not considered. In the best case, only definite Horn rules which can be expressed in DL can be

\(^{4}\)http://swat.cse.lehigh.edu/projects/lubm/
\(^{5}\)http://km.aifb.kit.edu/projects/owltools/
\(^{6}\)http://www.jessrules.com
considered. Furthermore, to the best of our knowledge, none of the current tools support the proposed incremental query answering functionality. On the other hand, our approach does not pose restrictions on the application-specific rules and requires no change in existing implementation.

Another line of research is towards incremental maintenance of materializations of ontological entailments. Materialization amounts to pre-computing and storing a set of implicit entailments, such that frequent and crucial queries to the ontology can be solved more efficiently. In [9], one such approach is presented which targets the DHL ontologies and follows the same translation method considered in this paper. After the mapping, the authors leverage on a declarative variant of the Delete and RE-Derive algorithm (DReD) [10] which is an incremental approach for the maintenance of intensional predicates in logic databases. In DReD, for a given logic program, a maintenance program is generated which maintains the materialization of the original program as changes occur in the extensional knowledge. Nevertheless, all maintenance operations are more expensive than the cost of evaluating a single query on the original program [9]. In fact, materialization is more suitable for read-dominant application domains. This approach further includes a comparatively redundant 'Rederive' phase which is necessary for truth maintenance. On the other hand, the Rete network employed in our method provides the sufficient infrastructure for the truth maintenance and serves as a 'minimal' solution to cater for the changes in the data.

VI. CONCLUSION AND FUTURE WORK

Ontology-based models of context information take a formal approach to foster interoperability, ease software development, and enable sound reasoning mechanisms. However, context data can be potentially dynamic which poses severe requirements on the efficiency of the query answering and thus reasoning methods. In this paper we presented an approach for incremental query answering in context aware systems. Our method can be applied to the Description Horn Logic (DHL) fragment of the Web Ontology Language (OWL). This fragment allows the translation of OWL schema statements to their equivalent definite Horn rules which can be later extended with application-specific rules. The rules corresponding to the OWL schema and the query are used to build the Rete network. This enables incremental maintenance of the query results as changes occur in the underlying knowledge base. To improve the performance, we provided a selection heuristic which efficiently chooses only rules which can affect the query results. As shown through experiments on the Lehigh University Benchmark (LUBM), our method yields acceptable results as new data are added to or retracted from the knowledge base. Using our method, incremental query answering functionality can be achieved without altering the current implementations of the OWL reasoners.

As a future work, we relax the current assumption that all the data necessary for the query answering are gathered in a central processing node. In fact, context sources are distributed and this assumption implies non-optimal use of network resources. In this regard, we believe it is important to develop distributed reasoning methods and query execution plans.

VII. ACKNOWLEDGMENT

This work is partially supported by project grant NRF2007IDM-IDM002-069 on Life Spaces from the IDM Project Office, Media Development Authority of Singapore.

REFERENCES