2012 International Conference on Software and Computer Applications (ICSCA 2012)
IPCSIT vol. 41 (2012) © (2012) IACSIT Press, Singapore

Towards Scalability Issue in Ontology-based Context-aware Systems

Cenzhe Zhu, Chen Guo, Jin Wang, and Teng Tiow Tay

ECE department, National University of Singapore

Abstract. Ontology is a promising tool to organize and represent context information. It arguably is superior to other context models in formality of semantics and the ability to deduce new facts. However, further usage of ontology-based systems is hindered by the slow reasoning speed of ontology engines and the monolithism of ontology knowledge bases. This paper proposes a fast and complete algorithm to extract sub-databases from the knowledge base for a given task. This algorithm is "fast" because it does not require an OWL (Web Ontology Language) reasoning process when filtering existing OWL triples and when synchronizing newly-added triples. We proved the completeness of the algorithm via an extensive analysis of the OWL semantics.

Keywords: context-aware system, ontology, OWL, scalability

1. Introduction

With the ubiquity of smartphones and smartphone-embedded sensors, an emerging concern on integrating the concept of context-awareness into everyday life is evident. Researchers are also constantly making explorations in building context-aware systems [2]. Among the various methods, ontology-based model for context-aware systems has its strength in distributed composition, strict semantics, the ability to be verified and reasoned and many more[1,4]. However, its drawbacks are obvious. Ontology reasoning is slow for real-time applications and the monolithic nature of knowledge base hinders the system's scalability.

This paper addresses the scalability problem of ontology-based context-aware systems by breaking the monolithism of knowledge bases, and thereby making the divide-and-conquer approach applicable.

If we do not classify tasks, each sub-database will require the whole database in order to process queries. Server cloning requires copies of the whole database, which can be very cumbersome; it also requires a large amount of real-time synchronization between all these copies as each update to a server should be pushed to all other servers. These two problems can be alleviated if we can further classify tasks (or, queries in our work), and allocate each category of tasks to a specific server, which primarily handles queries for this specific purpose. In the following, we use the word “application” interchangeable with a category of tasks or queries.

In classifying tasks, how to extract the back-end database for a specified task is nontrivial:

- It is hard to determine which are the necessary statements to be extracted for a functionality. One seemingly irrelevant statement may be useful for deducting relevant information.
- Generally, only raw statements but no deduced statements should be included in the extraction as such deduction can sometimes increase the size of the database exponentially.
- After extraction, how to keep the extracted database synchronized to the central one is also a problem.

This paper tackles these three problems by: 1. proposing a completeness-proved algorithm to determine which are the necessary triples to fulfil certain functionalities, and 2. migrating only existing triples (i.e., not including inferred triples) in order to control the size of sub-databases. Using the algorithm described in this paper, we can extract application-specific sub-databases from the whole knowledge base. It is also proved that the extracted sub-database can accommodate all queries from that specific application, which is also known as the completeness of the sub-database (or algorithm). The algorithm also covers the synchronization process. When an update of information is received at the server, it can be quickly decided (without going through an ontology reasoning process) whether it should be delivered to the sub-database.

The remainder of the paper is organized as such: Section 2 explains the extraction algorithm together with a recommendation on the decision of domain of discourse. The completeness of the algorithm is studied in section 3. We conclude in section 4.

2. Approaches

This section explains the algorithm to extract information from the whole knowledge base in order to answer a category of queries. It also covers the update procedure after the first extraction. The feature of this algorithm is that we use a set of filters to determine if a triple is relevant to the category of query. This algorithm is "complete" in the sense that this category of queries will receive a same response from the sub-database as it will receive from the intact knowledge base. It is assumed that the readers are familiar with the terminologies in *ontologies* and *description logic*.

Fig. 1 illustrates the topology of our system where AppServers hold sub-databases for a single application and Central Server holds the whole database.

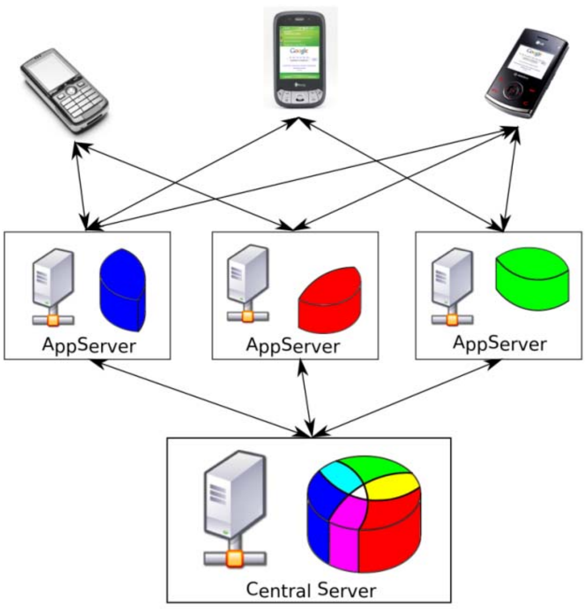

Fig. 1: System Structure

```
Let $\mathbb{NC}, \mathbb{E}, \mathbb{NE}$ be empty sets
loop until $\mathbb{C}, \mathbb{NC}, \mathbb{P}$ do not change
for all $C_i$ in $\mathbb{C}$
    add $P^{ANY \to ANY}$ in $\mathbb{P}$ if $P$ rdfs:domain $C_i$
    add $P^{ANY \to ANY}$ in $\mathbb{P}$ if $P$ rdfs:range $C_i$
    add $D_1, D_2$ in $\mathbb{C}$ if $D_1 \cup D_2 = C_i$
    add $D$ in $\mathbb{C}$ if $D = C_i \cap C_2$
    add $D$ in $\mathbb{NC}$ if $C_i$ is owl:complementOf $D$
    add $x_1, x_2, \ldots, x_l$ in $\mathbb{E}$ if $C_i = enum\{x_1, \ldots, x_l\}$
    add $D$ in $\mathbb{C}$ if $C_i$ is owl:equivalentClass to $D$
    add $D$ in $\mathbb{C}$ if $D \subset C_i$
    add $D$ in $\mathbb{C}$, add $P$ in $\mathbb{P}$ if $D = \forall P.C_i$
    add $P^{ANY \to y}$ in $\mathbb{P}$ if $C_i$ owl:hasValue $y$ on property $P$
end for
for all $C_i$ in $\mathbb{NC}$
    add $D$ in $\mathbb{NC}$ if $D = C_i \cup C_2$
    add $D_1, D_2$ in $\mathbb{NC}$ if $C_i = D_1 \cap D_2$
    add $D$ in $\mathbb{C}$ if $C_i$ is owl:complementOf $D$
    add $x_1, x_2, \ldots, x_l$ in $\mathbb{NE}$ if $C_i = enum\{x_1, \ldots, x_l\}$
    add $D$ in $\mathbb{NC}$ if $C_i$ is owl:equivalentClass to $D$
    add $D$ in $\mathbb{NC}$ if $C_i \subset D$
    add $D$ in $\mathbb{C}$ if $C_i$ is owl:disjointWith $D$
end for
for all $P$ in $\mathbb{P}$
    add $C$ in $\mathbb{C}$ if $C$ owl:hasvalue y on property $P$
    add $Q$ in $\mathbb{P}$ if $Q$ owl:inverseOf $P$
end for
end loop
```

Fig. 2: Filter Expansion

2.1. The Algorithm

We describe the algorithm in 3 phases---PREPARATION phase, SETUP phase, and UPDATE phase. In PREPARATION phase, we derive a set of filters from the characteristic of the application. In SETUP phase and UPDATE phase, these filters are applied to all existing triples and newly-updated triples.

PREPARATION phase:

Suppose we already know the domain of discourse of this application denoted by $(\mathbb{C}, \mathbb{P})$. $\mathbb{C}$ is the set of *Classes* $(C_1, C_2, \ldots, C_n)$, and each C_i denotes one *Class* in the vocabulary that is "relevant" to the application. $\mathbb{P}$ is the set of Properties $(P_1, P_2, \ldots, P_m)$, and P_i is a "relevant" *Property* P_i. Next subsection will discuss further on how to extract this domain of discourse of an application.

Then we construct a set of filters following the algorithm in Fig. 2.

In PREPARATION phase, we also need to pre-process the knowledge base by adding a unique class name for each of the anonymous classes.

SETUP phase:

The first step to extract a sub-database is to extract the vocabulary for it. So in SETUP phase, we first copy all TBox assertions from the whole knowledge base to sub-database. This amount can be reduced to only those "relevant" TBox triples to $\mathbb{C}$ and $\mathbb{P}$, but we simply leave it for now. Then a filtering procedure is carried out for all existing ABox triples in the knowledge base. Here we consider only 3 types of ABox assertions, leaving annotation assertions behind. I_1, I_2, I are *individuals*, C is a *class* name, P is a *property* name.

1. $(I, \text{rdf:type}, C)$. If C falls in the set $\mathbb{C}$ or $\mathbb{NC}$, copy the triple to sub-database.
2. (I_1, P, I_2). If the triple matches one element in set $\mathbb{P}$, copy it to sub-database.
3. $(I_1, \text{owl:sameAs}, I_2)$, $(I_1, \text{owl:differentFrom}, I_2)$. All triples of these two types are copied.

UPDATE phase:

When an update of triple is made to the knowledge base, it should be decided whether this update is relevant to the application under consideration, and therefore whether it should be delivered to the corresponding sub-database (server). This deciding procedure is the same as the filtering procedure for existing ABox assertions in SETUP phase.

2.2. Domain of Discourse

In this subsection, we give a means to determine the domain of discourse $(\mathbb{C}, \mathbb{P})$ given the description of an application.

The domain of discourse directly determines the size of a sub-database. It should be designed in such a way that the yielded sub-database is big enough to hold all required information, and is small enough to exclude useless information. Since we distinguish sub-databases by different applications, application developers should be responsible for the definition of domain of discourse. Nonetheless, we give a guideline as below:

1. First, enumerate all the possible query types that an application server might receive from its clients. This information can usually be taken from documentations from early development phases.
2. Assume queries from clients are in the format of *SPARQL* or other query languages. If not, abstract the interaction between application server and client as standard *SPARQL* queries.
3. For each query, there is a "*where*" clause. Include classes and properties that appear in this clause into the domain of knowledge. Take the finest category of classes or properties if one falls in a hierarchical structure. For example, a query may be "*SELECT ?x WHERE :John :LocatedIn ?x*". The extracted classes are "Man", "Location"; The extracted property is "LocatedIn". Note that "John" is of class "Person" and "Man", but the directly inherited class is taken into the domain of knowledge.

3. Proof of Completeness

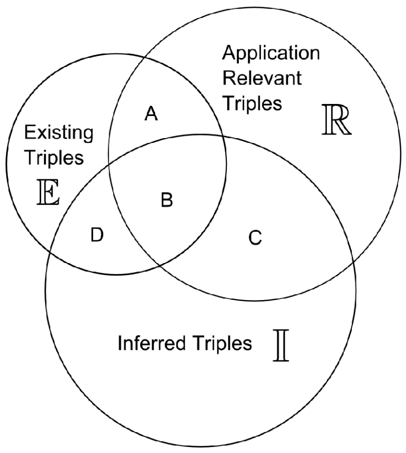

Fig. 3: The whole knowledge base is $\mathbb{E} + \mathbb{I}$. A complete extraction should at least comprise A+B+C

First we define the completeness of an extraction algorithm. In common sense, a complete extraction algorithm when applied to a specific type of queries, will extract a sub-database that can accommodate all queries of that type. As an illustration, Fig. 3 shows the relationship between the existing triples, inferred triples and required triples. The intersection between $\mathbb{E}$ and $\mathbb{I}$ indicates that some of the existing triples can be inferred by other existing triples. This is a sign of consistency of the knowledge base. With the Open World Assumption (OWA), some of the relevant triples may not fall in $\mathbb{E}+\mathbb{I}$. A complete extraction of the database should at least comprise or be able to infer triples in part A, B and C.

3.1. Inference rules given by OWL Semantics

Starting from the summary of OWL DL axioms and facts at [3], all OWL axioms and facts are listed and carefully analyzed. By translating all axioms and facts into Description Logic (DL) and First Order Logic (FOL), we get a better understanding of the semantics of OWL and how it can be applied to inferences. For the limitation of space, these intermediate translation tables are omitted with only the final listing of inference rules retained.

An inference procedure can use one or multiple axioms listed in the tables. But one thing we can assure is that the final outcome of the inference can only have 2 forms: "$C(i)$", or "$P(i_1, i_2)$". Any other statements are only intermediate results and must be combined with other axioms to deduce useful results.

The logic of the following proof is like this: We first list all possible combinations of axioms and facts that will produce $C(i)$ and $P(i_1, i_2)$. The outcome of the inference is placed in the LHS, and the pre-conditions are put in the RHS, with facts placed after axioms. Among the terms of facts in the RHS of the rules, copy those that are not included in the LHS to LHS and continue the process until no more updates to the listing. To kick start the procedure:

$$\begin{aligned}
C(x) &\Leftarrow C = D_1 \cup D_2,\quad D_1(x)\\
&\Leftarrow D = C \cap C_2,\quad D(x)\\
&\Leftarrow C = \neg D,\quad \neg D(x)\\
&\Leftarrow C = \text{enum}\{x_1,\ldots,x_l\},\quad x = x_i\\
&\Leftarrow C \equiv D,\quad D(x)\\
&\Leftarrow D \sqsubseteq C,\quad D(x)\\
&\Leftarrow D = \forall P.C,\quad D(y) \wedge P(y,x)\\
&\Leftarrow C = \exists P.\{i\},\quad P(x,i)\\
&\Leftarrow x = y,\quad C(y)\\
P(x,y) &\Leftarrow C = \exists P.\{y\},\quad C(x)\\
&\Leftarrow P \equiv Q^-,\quad Q(y,x)\\
&\Leftarrow P \equiv P^-,\quad P(y,x)\\
&\Leftarrow P^+ \sqsubseteq P,\quad P(x,z) \cap P(z,y)\\
&\Leftarrow x = z,\quad P(z,y)\\
&\Leftarrow y = z,\quad P(x,z)
\end{aligned}$$

Fig. 4: Primitive Inference Rules (first round)

$$\begin{aligned}
\neg C(x) &\Leftarrow D = C \cup C_2,\quad \neg D(x)\\
&\Leftarrow C = D_1 \cap D_2,\quad \neg D_1(x)\\
&\Leftarrow C = \neg D,\quad D(x)\\
&\Leftarrow C = \text{enum}\{x_1,\ldots,x_l\},\quad x \neq x_i\\
&\Leftarrow C \equiv D,\quad \neg D(x)\\
&\Leftarrow C \sqsubseteq D,\quad \neg D(x)\\
&\Leftarrow C \sqsubseteq \neg D,\quad D(x)\\
x_1 = x_2 &\Leftarrow D = \leqslant 1P.C,\quad D(y), P(y,x_1),\\
&\qquad C(x_1), P(y,x_2), C(x_2)\\
&\Leftarrow \top \sqsubseteq \leqslant 1P.\top,\quad P(y,x_1), P(y,x_2)\\
&\Leftarrow \top \sqsubseteq \leqslant 1P^-.\top,\quad P(x_1,y), P(x_2,y)
\end{aligned}$$

Fig. 5: Primitive Inference Rules (second round)

So far, the only form of fact on RHS that does not appear on LHS is $x_1 \neq x_2$, which shows inequality. However, inequality is not deductible. It can only be given in owl:differentFrom statements. So the procedure ends here with no more terms movable to LHS. Together we have 25 different forms of deduction rules.

3.2. Proof

After giving an exhaustive enumeration of inference rules, now we go back to prove that any triples in part C in Fig. 3 will be extracted by the algorithm we proposed.

For the simplicity of demonstration, assume $\mathbb{C} = \{C_T\}, \mathbb{P} = \{P_T\}$. The proof is done by contradiction.

Suppose there is at least one triple in part C that is not deducible by the sub-database using our algorithm, denoted by triple_C. It can be either of format $C_T(x_0)$ or $P_T(x_0, y_0)$. In either case, we can construct an inference tree for the triple in the context of the whole knowledge base. The root node of the tree is the triple in our discussion. At each branching, the children nodes are the axioms or facts that are used to deduce their parent node. Each deduction is atomic and cannot be further divided. The inference tree keeps expanding until all leaf nodes are existing axioms or facts that require no further deduction. So altogether, the tree manifests the inference procedure in producing triple_C.

If we treat all equalities and inequalities as if they are axioms (this is reasonable because in our algorithm, equalities and inequalities are not expanded but are directly copied to the sub-database just like axioms), the inference tree will have such features: each branching corresponds to an inference rule in Fig. 4 and 5; at each branching, one of the two children is an axiom, the other is a fact; all parent nodes are facts.

Because triple_C cannot be deduced in sub-database, there must be some leaf node presented in the inference tree that is absent from the extracted sub-database. Consider all possible cases:

1. The leaf node denotes an axiom. Hence, this piece of axiom is not present in the sub-database. This contradicts with the SETUP phase of the algorithm that all TBox assertions are directly copied.
2. The leaf node denotes a fact of equality or inequality. By OWL syntax, such equality or inequality can be directly given in knowledge base by owl:sameAs or owl:differentFrom assertions. However, these assertions are directly copied to the sub-database. And this yields contradiction.
3. The leaf node denotes a fact $C_m(x_m)$. Hence, C_m is not included in the set $\mathbb{C}$ in PREPARATION phase. Traversing from this leaf node back to the root node, we'll come across a sequence of facts $f_1, f_2, \ldots, f_{n-1}, f_n$, starting from $C_i(x_i)$ until triple_C. Because $C_i(x_i)$ does not fit in our algorithm's filters but triple_C does, along the sequence we can find the first fact f_i fits one of the filters while f_{i-1} does not. The inference rule $f_i \Leftarrow a_{i-1}, f_{i-1}$ shall be one of the rules in \ref. Now that there is a bijection between the filter expanding procedure and the rules in \ref, the filter that can match f_i is sure to be expanded to a filter that can match f_{i-1}. This contradicts with the knowledge that f_{i-1} does not fit in any filters in the algorithm.
4. The leaf node denotes a fact $P_m(x_m, y_m)$. The same logic as case 3 will yield contradiction.

In all 4 cases, we have contradictions. Thus, the initial supposition is false, and our algorithm is complete.

4. Conclusion and Future Works

We have proposed an extraction algorithm to break the monolithism of ontology-based databases. This enhances scalability of ontology-based context-aware systems in the sense that divide-and-conquer becomes available. Using applications as the extraction boundary, we are able to design a loosely-coupled network system to manage context information. The extraction algorithm is proved to be complete and the algorithm avoids OWL reasoning when updating any new triples. We are currently building a new test bench specially designed for context-aware systems on real use cases. A more quantitative experiment is in process. A preliminary experiment has shown that our algorithm requires at least 90% time less than a trivial extraction algorithm. We are also tuning the algorithm, exchanging the strict completeness for more flexibility and performance benefits.

5. References

[1] M. Baldauf, S. Dustdar, and F. Rosenberg. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4):263–277, 2007.

[2] J. Hong, E. Suh, and S.J. Kim. Context-aware systems: A literature review and classification. *Expert Systems with Applications*, 36(4):8509–8522, 2009.

[3] Peter F. Patel-Schneider and Ian Horrocks. Owl web ontology language semantics and abstract syntax section 2. abstract syntax, *http://www.w3.org/TR/2004/REC-owl-semantics-20040210/syntax.html*, February 2004.

[4] T. Strang and C. Linnhoff-Popien. A context modeling survey. In *Workshop on Advanced Context Modelling, Reasoning and Management as part of UbiComp*, 2004.