A platform for distributing and reasoning with OWL-EL knowledge bases in a Peer-to-Peer environment

Alexander De Leon[1], Michel Dumontier[1,2,3]

[1] School of Computer Science
[2] Department of Biology
[3] Instititute of Biochemistry
Carleton University, 1125 Colonel By Drive, K1S 5B6, Ottawa, Canada
adlbatti@scs.carleton.ca, michel_dumontier@carleton.ca

Abstract. Memory exhaustion is a common problem in tableau-based OWL reasoners, when reasoning with large ontologies. One possible solution is to distribute the reasoning task across multiple machines. In this paper, we present, as preliminary work, a prototypical implementation for distributing OWL-EL ontologies over a Peer-to-Peer network, and reasoning with them in a distributed manner. The algorithms presented are based on Distributed Hash Table (DHT), a common technique used by Peer-to-Peer applications. The system implementation was developed using the JXTA P2P platform and the Pellet OWL-DL reasoner. It remains to demonstrate the efficiency of our method and implementation with respect to stand alone reasoners and other distributed systems.

1 Introduction

Scalability is the main technological challenge faced by OWL-DL reasoners. This issue is a natural consequence of the computational complexity of reasoning problems. The tableau algorithm, used by most OWL-DL reasoners, is known to be NExpTime-complete and can create data structures of double exponential size of the input [6]. In other words, the algorithm is not only CPU intensive but also requires a large amount of memory for the case of large ontologies. This paper explores the idea of implementing a distributed OWL reasoner using a peer-to-peer (P2P) approach. The goal is to distribute the tableau algorithm such that it can utilize the combined resources of the P2P network, and therefore mitigate the memory issues that arise when reasoning with large ontologies on a single machine.

As part of the OWL 2 recommendation, different fragments are proposed which trade expressivity in favor of efficient reasoning. These fragments have been named OWL profiles and each of them defines a set of syntactic restrictions of the full OWL 2 language. Each profile targets a different type of application. The OWL-EL profile is a subset of OWL-DL based on the $\mathcal{EL}++$ light weight description logic [4]. This profile is recommended for ontologies with a large

Proceedings of OWL: Experiences and Directions 2009 (OWLED 2009), Rinke Hoekstra, editor.
http://www.webont.org/owled/2009

number of classes and properties and where classification and instance checking are the primary reasoning task of interest. Reasoning problems in the EL fragment become polynomial with respect to the size of the ontology [1] . The implementation presented in this paper supports only ontologies in the OWL-EL profile. The reason for choosing this profile is that the tableau algorithm for EL is simpler than it would otherwise be if supporting the full OWL language. This simplification arises from the fact that EL eliminates the sources of non-determinism such as concept disjunction, negation of complex concepts, and universal restrictions.

The system presented in this paper makes use of the JXTA[4] P2P framework and the Pellet[5] reasoner. JXTA is used as the underlying platform for peer discovery and communication. Pellet is a popular sound and complete reasoner for OWL-DL [8]. The distributed algorithm presented was implemented as an extension of Pellet. The reason for choosing these technologies is that they are both publicly available as open source and they were both implemented in Java.

2 Software Architecture Overview

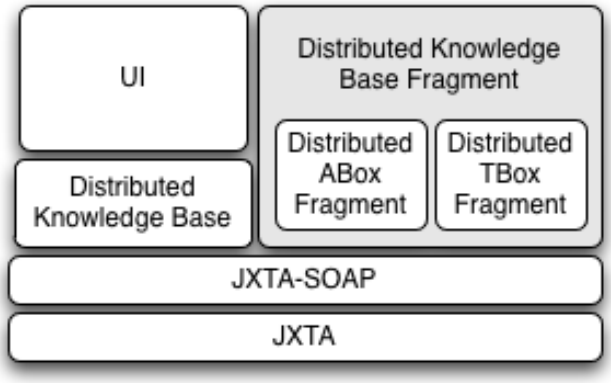

Fig. 1: Peer's Software Layers

Each peer is implemented in Java using the JXTA framework. JXTA sits at the bottom of the peer's software stack (Fig. 1), and provides the infrastructure for peers to discover themselves and establish communication channels. The next layer is implemented with a modified version of the JXTA-SOAP[6] library. This

[4] https://jxta.dev.java.net/
[5] http://clarkparsia.com/pellet
[6] https://soap.dev.java.net/

layer allows publishing, discovery and invocation of SOAP services on top of the JXTA network. The *Distributed Knowledge Base Fragment* component is responsible for managing the subset of the knowledge base's axioms that correspond to this peer. Multiple of these fragments can co-exists inside a peer, given that the peer can be a collaborator in the distribution of multiple independent knowledge bases (Fig. 2). The *Distributed Knowledge Base Fragment* implements a SOAP service interface which is advertised on the P2P network. The *Distributed Knowledge Base* discovers the *Distributed Knowledge Base Fragment* of other peers and uses their SOAP interface to communicate with them. This component coordinates the distribution of the knowledge base across the participating peers and the execution of reasoning tasks. The software interface of *Distributed Knowledge Base* is identical to that of the Pellet's KnowledgeBase[7] class. Therefore, higher-level components can interact with the *Distributed Knowledge Base* in the same way they will do when using Pellet and without been aware of the distributed nature of the reasoner. The user interface (UI) allows human users to interact with the *Distributed Knowledge Base* component. Currently there is a command line interface implemented, however an additional web based GUI is envisioned for the next generations of the software.

3 Distributed Knowledge Base

In the P2P system described by this paper, each peer can act as a reasoner, allowing users to use the peer to load an ontology and reason with it in the same way they would do when using a non-distributed OWL reasoner. This peer becomes responsible for coordinating the distribution of the knowledge base and the reasoning task. To load an ontology, the coordinator peer will try to find a specified number of peers which will be responsible for a subset of the knowledge base. If after a specified amount time, not all the requested peers have been found on the P2P network, the loading process continues using only those peers that have been discovered so far. Each ontology is loaded within the context of a session, allowing a peer to participate in multiple reasoning tasks independently of each other (Fig. 2) .

Individual axioms from the ontology are distributed across the participating peers following a distributed hash-table approach [9]. Each participating peer has a unique identifier provided by the JXTA platform. This identifier is converted into a numerical hash using the SHA1 algorithm [5]. Similarly, for a given class, property or individual, a numerical hash can be constructed by applying the SHA1 function to its URI. Each peer is responsible for a fragment of the entire knowledge base. This fragment contains:

1. the subset of the base concepts whose URI hash is numerically closest to the hash of the peer's identifier;
2. the subset of terminological axioms of the form $equivalentTo(C, D)$ and $subClassOf(C, D)$ for which the URI hash of C is numerically closest to

[7] org.mindswap.pellet.KnowledgeBase

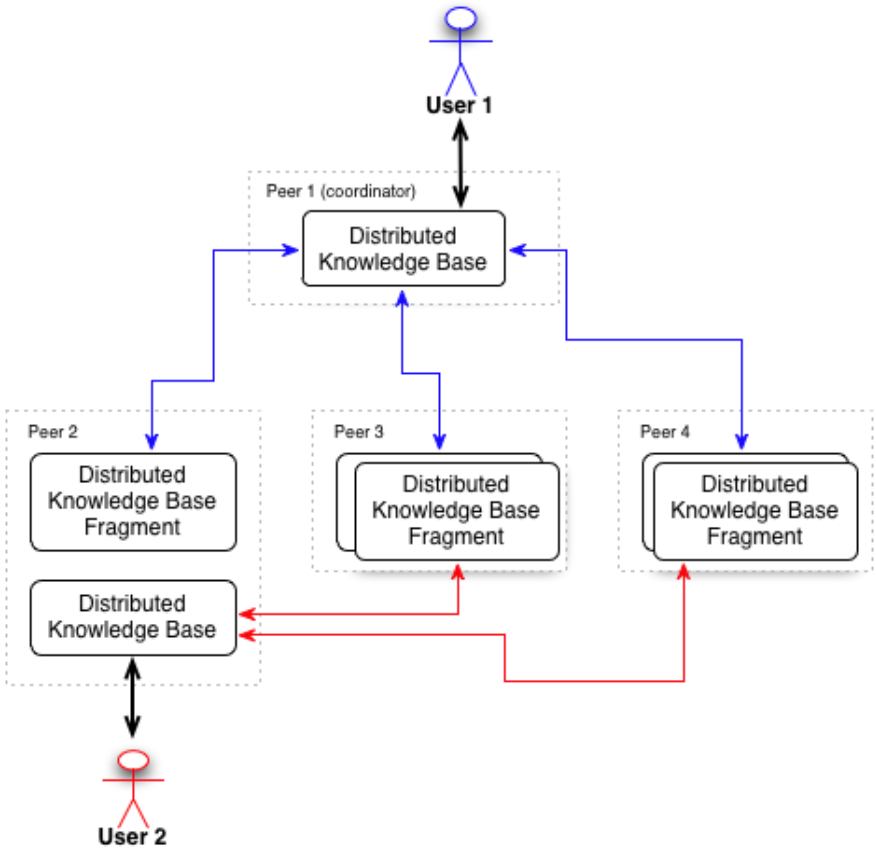

Fig. 2: Multiple users using the P2P reasoning services from different peers

the peer's id hash (note that general inclusion axioms are not supported therefore C is always a named concept);

3. all axioms about properties (e.g. $transitiveProperty(P)$, $functionalProperty(P)$, ect);
4. all individual assertions of the form $C(a)$ and $R(a, b)$ where the URI hash of a is numerically closest to the peer id hash.

Table 1 presents a small knowledge base about Vehicles. Using this as an example and assuming a network of two peers, the knowledge base distribution is shown in Table 2. Axioms about properties (RBox) are replicated in every node. The number of these axioms is usually much smaller than the number of axioms in the TBox and ABox. For this reason, it is preferable to have them available locally at each peer to reduce the amount of peer messaging during the execution of the reasoning algorithm.

Table 1: Complete Knowledge Base

Axiom	**Hash key**
TBox	
$Vehicle \sqsubseteq \top$	Vehicle=50
$Engine \sqsubseteq \top$	Engine=190
$Bicycle \sqsubseteq Vehicle$	Bicycle=220
$Car \sqsubseteq Vehicle \sqcap \exists hasPart(Engine)$	Car=120
$Automobile \equiv Car$	Automobile=170
RBox	
$transitiveProperty(hasPart)$	hasPart=33
ABox	
$Automobile(a)$	a=99
$Bicycle(b)$	b=280
$Engine(e)$	e=201
$hasPart(a, e)$	a=99

Table 2: Distributed Knowledge Base

(a) Peer 1

Peer 1 (hash key = 100)
TBox
$Vehicle \sqsubseteq \top$
$Car \sqsubseteq Vehicle \sqcap \exists hasPart(Engine)$
RBox
$transitiveProperty(hasPart)$
ABox
$Automobile(a)$
$hasPart(a, e)$

(b) Peer2

Peer 2 (hash key = 200)
TBox
$Engine \sqsubseteq \top$
$Bicycle \sqsubseteq Vehicle$
$Automobile \equiv Car$
RBox
$transitiveProperty(hasPart)$
ABox
$Bicycle(b)$
$Engine(e)$

4 Reasoning with a Distributed Knowledge Base

This section presents how the system presented in this paper performs two important reasoning tasks: concept satisfiability and ABox consistency. These reasoning problems form the base of other reasoning tasks such as subsumption and instance checking.

4.1 Concept satisfiability

Usually we want to know if a concept C is satisfiable w.r.t. a TBox $\mathcal{T}$. *Unfolding* is a well known technique to allow satisfiability testing of concepts independent from the TBox. The idea is to produce a new concept C', such that C *is satisfiable*

w.r.t. $\mathcal{T}$ *iff* C' *is satisfiable*. The concept C' is called the *expansion* of C, and it is obtained recursively from C by replacing each non base symbol A in C by the concept D, where D is the expansion of A. A detailed definition of this technique is given in [3].

For example, given the following TBox:

$$Female \equiv \neg Male$$
$$Parent \equiv \exists hasChild.Person$$
$$Mother \equiv Female \sqcap Parent$$

The expansion of $Mother$ is $\neg Male \sqcap \exists hasChild.Person$. This expansion is now expressed using only base symbols and, therefore, it is independent from the TBox. The satisfiability of the expansion can be tested using traditional symbolic refutation. In order to use *unfolding* with a TBox containing inclusion axioms, like the TBox from Table 1, the TBox needs to be normalized to convert all inclusions axioms to definition axioms. This normalization process is explained in [3].

In the case of distributed knowledge bases, the procedure presented in Algorithm 1 is used to obtain the expansion of a concept. When a concept needs to be unfolded, the peer responsible for the concept will be asked. The responsible peer will first unfold the concept locally (using the underlying Pellet system), and then each remote concept in the result will be replaced by its expansion provided by the remote peer.

Algorithm 1 DISTRIBUTEDUNFOLD(c)

Require: An concept name c.
Ensure: A list of concepts whose intersection form the expansion of c

```
1: if isRemote(c) then
2:     peer = getResponsiblePeer(c)
3:     return peer.DISTRIBUTEDUNFOLD(c)
4: else
5:     unfolding = localUnfold(c)
6:     for all d in unfolding do
7:         if isRemote(d) then
8:             unfolding.remove(d)
9:             peer = getResponsiblePeer(d)
10:            remoteUnfolding = peer.DISTRIBUTEDUNFOLD(d)
11:            unfolding.addAll(remoteUnfolding)
12:        end if
13:    end for
14:    return unfolding
15: end if
```

4.2 ABox consistency

An Abox $\mathcal{A}$ is consistent with respect to a TBox $\mathcal{T}$ iff there exists an interpretation which is a model of $\mathcal{A}$ and $\mathcal{T}$. Similarly with concept satisfiability, if all concepts have been expanded (unfolded), then we can perform ABox consistency checking without considering the full TBox.

In order to reason with the ABox, we need to consider two other cases where peers need to exchange information: resolving the class membership of a remote individual and obtaining the set of edges that are connected to a remote individual. The former case is handled by asserting each remote individual to be a member of the class $REMOTE(i)$, where i is the name of the individual. When the local Pellet algorithm tries to unfold $REMOTE(i)$, our distributed unfolding implementation handles this by going to the responsible peer for i and asking it for the unfolded set of classes that i is a member of. Therefore, in the resulting expanded ABox, $REMOTE(i)$ will be replaced by the actual base concepts that the individual belongs to. In order to handle the latter case, the implementation of the *some-rule* (i.e. the one that handles existential restrictions) of the tableau algorithm has been modified. The modified part checks if the individual is remote, and if it is so, then it requests information about its edges from the remote peer.

By handling these two cases, the consistency of the entire distributed knowledge base can be done by checking the consistency of each fragment simultaneously. This is true only under the assumption that we are dealing with ontologies in the OWL-EL profile. If we had more expressive ontologies, with universal restrictions and inverse properties, for example, then the order in which the fragments are checked matters, since the expansion of one peer's fragment can introduce changes to the fragments of other peers. Furthermore, more peer interaction cases will need to be handled in addition to the two cases described above.

5 Conclusion and Future Work

We described a basic P2P OWL reasoner implementation, which derives from the integration of the JXTA platform and the Pellet reasoner. The techniques presented in the paper target OWL ontologies within the OWL-EL profile, given that the constructs in this subset of OWL 2 have simpler reasoning algorithms. We described how the system deals with the distribution of two basic reasoning problems: concept satisfiability and ABox consistency checking. The soundness and completeness of these techniques are consequence to the fact that the system reuses the tableau implementation of the Pellet reasoner, which has been previously shown to sound and complete [8]. However, further theoretical and experimental analysis of these techniques is desirable for future work. It remains to evaluate the results of our prototypical implementation and compare them to other systems such as the CEL reasoner [2] and KAON2 [7].

For the next steps of this project, we want to implement query answering using the instance checking reasoning task. This allows for querying the dis-

tributed knowledge base by providing a new concept description and checking at each peer which individuals can be classified as members of the query class. An area of further study, is that of using ontology modularization techniques for distributing the knowledge base such that axioms which are semantically related can reside in the same peer. It is suspected that this kind of semantic distribution will reduce the number of inter-peer messages during the executing of the reasoning algorithms.

Handling peer failure is an important area which has not yet been addressed. We suggest using a group of peers, instead of a single peer, to handle a particular fragment of the ontology. Although this may not been considered a complete solution, it greatly reduce the severity of the issue.

References

1. OWL 2 Web Ontology Language Profiles, `http://www.w3.org/TR/owl2-profiles/`.
2. F. Baader, C. Lutz, and B. Suntisrivaraporn. CEL—a polynomial-time reasoner for life science ontologies. In U. Furbach and N. Shankar, editors, *Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR'06)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 287–291. Springer-Verlag, 2006.
3. Franz Baader. *The description logic handbook: theory, implementation, and applications*. Cambridge University Press, Cambridge, 2nd ed edition, 2007.
4. Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the el envelope further. In *OWLED08DC*, 2008.
5. D. Eastlake 3rd and P. Jones. US Secure Hash Algorithm 1 (SHA1). RFC 3174 (Informational), September 2001. Updated by RFC 4634.
6. Ian Horrocks and Ulrike Sattler. A tableau decision procedure for $\mathcal{SHOIQ}$. *J. Autom. Reason.*, 39(3):249–276, 2007.
7. B. Motik. Practical DL Reasoning over large ABoxes with KAON2. Available at: `http://kaon2.semanticweb.org/`. (2006).
8. Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical owl-dl reasoner. *J. Web Sem.*, 5(2):51–53, 2007.
9. Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149–160, New York, NY, USA, 2001. ACM.