Matchmaking Using Instance Store: Some Preliminary Results

Lei Li and Ian Horrocks
University of Manchester, Manchester, UK
Email: {lil, horrocks}@cs.man.ac.uk

Abstract

An important objective of the Semantic Web is to make Electronic Commerce interactions more flexible and automated. To achieve this, standardization of ontologies, message content and message protocols will be necessary.

In this paper we investigate how Semantic and Web Services technologies can be used to support service advertisement and discovery in e-commerce. In particular, we present some preliminary results of a service matchmaking prototype which uses the instance store to retrieve ontology based individual descrption and show the difference with using the traditional DL reasoner.

1 Motivation

The Semantic Web requires that data be not only machine readable (just like the Web nowadays does), it also wants the data to be machine understandable. To quote Tim Berners-Lee, the director of the World Wide Web consortium (W3C), and prime architect of the Semantic Web:

> The semantic web goal is to be a unifying system which will (like the web for human communication) be as un-restraining as possible so that the complexity of reality can be described [4].

With a Semantic Web, it will be easy to realise a whole range of tools and applications that are difficult to handle in the framework of the current web. Examples include knowledge-repositories, search agents, information parsers, etc. Moreover, the developers of end user applications will not need to worry about how to interpret the information found on the Web, as ontologies will be used to provide vocabulary with explicitly defined and machine understandable meaning [10].

One important Semantic Web application area is e-commerce. In particular, a great deal of attention has been focused on semantic *web services*, the aim of which is to describe and implement web services so as to make them more accessible to automated agents. Here, ontologies can be used to describe services so that agents (both human and automated) can advertise and discover services according to a semantic specification of functionality (as well as other parameters such as cost, security, etc.) [13].

Moreover, if applications are to exchange semantic information, they will need to use common ontologies. In this paper, we present a case study of an e-commerce application in which a predefined service ontology is used to provide the vocabulary for service descriptions. These descriptions are used in a match-making prototype, i.e., a repository where agents can advertise and search for services that match some semantic description.

With the possibilities opened by e-commerce, the number of potential service advertisements could be huge, millions of them might need to be examined to filter out relevant ones. However, loading and reasoning such large amount of ontology based services seems an intractable task for ABox reasoning of the DL reasoners nowadays. To resolve this problem, a simple DL instance store idea was proposed in the IMG group at the University of Manchester. It aims at weaker but probably sufficient reasoning functionalities to support a number of applications where individual objects are to be described in terms of concept terms from a DL model and then subsequently retrieved using concept descriptions queries[3].

In our prototype, we used JADE [9] as the agent platform for our prototype and we used the instance store in order to manage all the service advertisements and service requests. The Racer DL reasoner is used behind the instance store to compute semantic matches between service advertisements and service requests. Finally, we carry out a performance analysis using this prototype in order to discover if the approach is likely to be feasible in large scale web applications.

2 Backgroud

2.1 Description Logic

Description Logics are a well-known family of knowledge representation formalisms. They are based on the notion of concepts (unary predicates, classes) and roles (binary relations), and are mainly characterized by constructors that allow complex concepts and roles to be built from atomic ones [8]. A DL reasoner can check whether two concepts subsume each other [7].

We assume the reader to be familiar with DLs, a detailed discussion of DLs is referred to [1].

2.2 Precompletion

One of the techniques for tackling the ABox reasoning is the precompletion algorithm, this technique is to split the KB satisfiability algorithm in two parts. In the first part all the information implicit in the role assertions is made explicit, generating what we call a "precompletion" of the knowledge base, then a terminological reasoner is used for verifying the consistency of the concept assertions for each individual name [5, 6].

The work presented in [12] extended the precompletion algorithm for Knowledge Base satifiability to the DL language $\mathcal{SHF}$. The main advantages of this approach are its ability of using existing highly optimised TBox reasoners, and the fact that it may be able to handle very large ABox by partitioning them into disconnected parts.

However, the logic concerned in [12] is a subset of that required by Semantic Web knowledge representation languages such as OWL, and it is not easy to see how the technique can be extended to deal with more expressive languages. Besides, as we stated in the motivation, we will be facing a huge amount of service advertisements which might be more challenging for the traditional ABox reasoning. Therefore, we chose an alternative approach — the instance store.

2.3 Instance Store

A instance store provides weaker functinality than a full ABox, but which is probably sufficient to support a number of applications where individual objects are to be described and then subsequently retrieved using concept descriptions.

The instance store is implemented using a conventional database and a reasoner, the database stores the relationships between the individuals and their concept descriptions. By using a combination of query against the database and subsumption/classfication requests to the reasoner, the instance store can answer queries such as "Retrieval[1]" and "Realization[2]" which used to require the full ABox reasoning. For a complete description of instance store, the insterested reader is referred to [3].

3 Matchmaking Using Instance Store

3.1 Service Ontology

Service description ontologies will have an important role to play in our work, so we have designed a domain-specific sample ontology about the sales of computers

[1]Which individuals occurring in the assertions are instances of concept description C?

[2]Given an individual i, what are the most specific concepts C in the TBox that i is an instance of?

in order to achieve matching at the semantic level between various parties. In our prototype, we used the OilEd [11] ontology editor to build DIG [2] formated ontologies. For the purpose of clarity and compactness, however, in this paper we will use the DL notions in place of the DIG syntax.

In our ontology, we use the `ServiceProfile` as a common superclass, so the other categorized service can be expressed as:

$$
\begin{array}{rcl}
ServiceProfile & \sqsubseteq & \top \\
SmallSerivce & \equiv & ServiceProfile \sqcap (PC \sqcap \leq_{200} unitQuantity) \\
MediumService & \equiv & ServiceProfile \sqcap (PC \sqcap >_{200} unitQuantity \sqcap <_{500} unitQuantity) \\
LargeService & \equiv & ServiceProfile \sqcap (PC \sqcap \geq_{500} unitQuantity) \\
AffordableService & \equiv & ServiceProfile \sqcap (PC \sqcap \leq_{400} unitPrice) \\
AdvancedService & \equiv & ServiceProfile \sqcap (PC \sqcap >_{400} unitPrice \sqcap <_{600} unitPrice) \\
CuttingEdgeService & \equiv & ServiceProfile \sqcap (PC \sqcap \geq_{600} unitPrice) \\
AMDOnlyService & \equiv & ServiceProfile \sqcap (PC \sqcap \forall hasCpu.AMDCpu) \\
IntelOnlyService & \equiv & ServiceProfile \sqcap (PC \sqcap \forall hasCpu.IntelCpu) \\
LargeMemService & \equiv & ServiceProfile \sqcap (PC \sqcap \geq_{512} memorySize) \\
 & & \\
LargeAMDService & \sqsubseteq & LargeService \sqcap AMDOnlyService \\
SmallAdvancedService & \sqsubseteq & SmallService \sqcap AdvancedService \\
MediumAdvancedIntelService & \sqsubseteq & MediumService \sqcap AdvancedService \sqcap IntelOnlyService \\
\ldots & &
\end{array}
$$

Suppose that we want to define an advertisement as an instance of the following service description:

- the provider is a large service provider;
- items are PCs and the processor brand must be Intel;
- the unit price must be less than 500.

In DL notaion, this advertisement `Advert1` can be written as:

$$
\begin{array}{rcl}
Advert1 & \in & LargeService \sqcap \\
 & & IntelOnlyService \sqcap \\
 & & ServiceProfile \sqcap (PC \sqcap <_{500} unitPrice)
\end{array}
$$

3.2 Matching Definition

Matchmaking is defined as a process that requires a repository host to take a query or advertisement as input, and to return all advertisement which may potentially satisfy the requirements specified in the input query or advertisement. Formally, this can be specified as:

Let α be the set of all advertisements in a given advertisement repository. For a given query or advertisement, Q, the matchmaking algorithm of the repository host returns the set of all advertisements which are compatible, $matches(Q)$:

$$matches(Q) = \{A \in \alpha | compatible(A, Q)\}$$

This compatible stands if `D2` is subsumed by `D1`[3]:

$$compatible(D1, D2) \Leftrightarrow (D2 \sqsubseteq D1)$$

For example, consider the following query:

$$Query1 \ \in \ LargeService \sqcap \ IntelOnlyService \sqcap \ AffordableService$$

This `Query1` is compatible with `Advert1` in Section 3.1. Formally:

$$Advert1 \in matches(Query1)$$

4 Prototype Implementation

In this section we describe the implementation of a multi-agent system including matchmaking, advertising and querying agents. Some issues, however, such as security (e.g., fraud), have not been taken into account, as they were not considered relevant to our purpose: the investigation of ontology based service description and a DL based matchmaking service.

4.1 Abstract Roles

To test the usability of service descriptions and matchmaking in the Semantic Web, we will introduce a scenario in which agents play a variety of roles. They are:

- **Host** manages the instance store, and performs the matching function by communicating with the instance store.
- **Advertiser** publishes advertisements to the host, and modifies, withdraws and browses advertisements stored in the instance store.
- **Seeker** sends a query to the host, and gets the matched advertisements back.

All these three kinds of abstract roles might be played by the same entity at different times or even at the same time, e.g., an information broker is an Advertiser and a Seeker at the same time. With this abstract definition, we can cover different types of matchmaking systems by adding one or more roles to the concrete entity in the real system.

[3]Note that, this compatible definition may vary from different applications, one could define his own in accordance to the particular requirement, e.g., an intersection relationship could also be compatible.

4.2 Functionalities

The matchmaking service provides three kinds of functionalities: advertising a service, querying a service, withdrawing the published service, modifying the published service and browsing advertised services in the repository.

4.2.1 Advertising

The Advertiser publishes to the Host the advertisement id together with a service description of what it is providing or seeking for. This description captures the relevant features of the service which will be used in matchmaking.

4.2.2 Querying

The Seeker can submit a query to find relevant advertisements among the currently available ones. By adding constraints over aspects that the Seeker is interested in, the query can be used to filter out irrelevant advertisements. Currently, only volatile query has been implemented in our prototype, persistent query could be added at a later stage.

4.2.3 Browsing

The Host offers the functionality of browsing the currently available advertisements. It maintains an instance store, where published advertisements are stored. In finding out about advertised services, browsing parties can make use of this information to tune the advertisements that they will submit in turn, so as to maximize the likelihood of matching.

4.3 Agents

We chose JADE as the agent platform, the goal of JADE being to simplify the development of multi-agent systems while ensuring standard compliance through a comprehensive set of system services and agents in compliance with FIPA specifications. The benefit of JADE is that we can concentrate on the agent functionalities and leave other things, like communication between agents, to the platform.

Three kinds of agents have been implemented:

- **HostAgent** has responsibility to initialize the instance store using assigned ontologies and a HTTP service enabled DL reasoner. This is the core component of the system, and its operation is described in more detail in Section 4.4 below.
- **AdvertiserAgent** publishes advertisement to the HostAgent and browses what was stored in the instance store.

- **SeekerAgent** publishes a request to the HostAgent. It also has the browse functionality.

4.4 Matchmaking

At the beginning of the matchmaking process, the HostAgent initializes the instance store with the service ontology described in Section 3.1, which the embeded Racer system will use to compute the subsumption relations between advertisements and requests throughout the whole matchmaking process.

When it receives an advertisement, the HostAgent asserts it and stores all the relevant information in the instance store.

When it receives a request, the HostAgent uses the instance store system to retrieve all the compatible advertisements. If the retrieval is not empty, the matched result for this request is returned to the seeker agent.

5 Evaluation

Processing.

6 Conclusions and Future work

In this paper we have introduced some preliminary results of our service matchmaking using instance store. We believe that these results are useful for searching web services based on a huge scale. The main features of our work can be summarized as follows.

The ontology based service descriptions can benefit the service discovery in e-commerce. This argument is supported by our design and implementation of a prototype matchmaker which uses an instance store to match service advertisements and requests based on the semantics of ontology based service descriptions. By representing the semantics of service descriptions, the matchmaker enables the behaviour of an intelligent agent to approach more closely that of a human user trying to locate suitable web services.

Instance store can be one possible solution when facing millions of advertisements in the realistic e-commerce applications.(wait for the evaluation).

References

[1] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider, editors. *The Description Logic Handbook*. 2002.

[2] Sean Bechhofer. The dig description logic interface: Dig/1.1. In *Proceedings of the 2003 Description Logic Workshop (DL 2003)*, 2003.

[3] Sean Bechhofer, Ian Horrocks, and Daniele Turi. Instance store: Database support for reasoning over individuals. In *Proceedings of the 2003 Description Logic Workshop (DL 2003)*, 2003.

[4] Tim Berners-Lee. The semantic web as a language of logic, 1998.

[5] F.M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Deduction in concept languages: From subsumption to instance checking. *Journal of Logic and Computation*, 4(4):423–452, 1994.

[6] Bernhard Hollunder. Consistency checking reduced to satisfiability of concepts in terminological systems. *Annals of Mathematics and Artificial Intelligence*, 18:95–131, 1996.

[7] I. Horrocks and P. F. Patel-Schneider. Comparing subsumption optimizations. In E. Franconi, G. De Giacomo, R. M. MacGregor, W. Nutt, C. A. Welty, and F. Sebastiani, editors, *Collected Papers from the International Description Logics Workshop (DL'98)*, pages 90–94. CEUR, May 1998.

[8] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In H. Ganzinger, D. McAllester, and A. Voronkov, editors, *Proceedings of the 6th International Conference on Logic for Programming and Automated Reasoning (LPAR'99)*, number 1705 in Lecture Notes in Artificial Intelligence, pages 161–180. Springer-Verlag, 1999.

[9] http://jade.cselt.it/.

[10] D. L. McGuinness. Ontological issues for knowledge-enhanced search. In *Proc. of FOIS*, Frontiers in Artificial Intelligence and Applications. IOS-press, 1998.

[11] Oiled portal, http://oiled.man.ac.uk/.

[12] S. Tessaris. *Questions and answers: reasoning and querying in Description Logic*. PhD thesis, University of Manchester, 2001.

[13] D. Trastour, C. Bartolini, and C. Preist. Semantic web support for the business-to-business e-commerce lifecycle. *HP Labs Technical Report HPL-2002-3*, 2002.