A Software Framework for Matchmaking Based on Semantic Web Technology

Lei Li and Ian Horrocks

ABSTRACT: The semantic Web can make e-commerce interactions more flexible and automated by standardizing ontologies, message content, and message protocols. This paper investigates how semantic and Web Services technologies can be used to support service advertisement and discovery in e-commerce. In particular, it describes the design and implementation of a service matchmaking prototype that uses a DAML-S based ontology and a description logic reasoner to compare ontology-based service descriptions. By representing the semantics of service descriptions, the matchmaker enables the behavior of an intelligent agent to approach more closely that of a human user trying to locate suitable Web services. The performance of this prototype implementation was tested in a realistic agent-based e-commerce scenario.

KEY WORDS AND PHRASES: Matchmaking, ontologies, semantic Web, Web services.

The semantic Web requires not only that data be machine-readable (just as the Web does nowadays), but that they 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 that will (like the Web for human communication) be as un-restraining as possible so that the complexity of reality can be described. [6]

With a semantic Web, it will be easy to realize a range of tools and applications that are difficult to handle in the framework of the current Web. Examples include knowledge repositories, search agents, and information parsers.

Moreover, the developers of end-user applications will not need to worry about how to interpret the information found on the Web, because ontologies will be used to provide vocabulary with explicitly defined and machine-understandable meaning [18].

Electronic commerce is an important semantic Web application area. 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 and security) [25].

The first author thanks members of the Intelligent Enterprise Technology Laboratory (IETL) at HP Labs, especially David Trastour and Claudio Bartolini, for their kind help and support. The first author was funded by the Departmental Scholarship in the Department of Computer Science at the University of Manchester.

As a first step in realizing the semantic Web, new standards for defining and using ontologies are already being developed. RDF, under development by the W3C RDF Core working group, is a Web markup language that provides basic ontological primitives [7]. DAML+OIL, an ontology language that extends RDF with a much richer set of primitives (e.g., boolean operators and cardinality constraints), is now the basis for the W3C Web Ontology Language working group's development of the OWL ontology language standard [4, 9].

If applications are to exchange semantic information, they will need to use common ontologies. One such ontology, written in DAML+OIL and designed for describing Web services, is the DAML-S ontology [24]. This paper presents a case study of an e-commerce application in which the DAML-S service ontology is used to provide the vocabulary for service descriptions.[1] These descriptions are used in a matchmaking prototype— a repository where agents can advertise and search for services that match a semantic description. The agent platform for the prototype was JADE, and the RACER DL reasoner was used to compute semantic matches between service advertisements and service requests [5, 10]. The paper illustrates some difficulties both in the application of the DAML-S ontology and in the use of the DL reasoner, and shows how these were overcome in the prototype implementation. Finally, the prototype is subjected to a performance analysis in order to discover whether the approach is likely to be feasible in large-scale Web applications.

Background

Ontology Languages

As already mentioned, ontologies play a key role in the semantic Web by providing vocabularies that applications can use to understand shared information.

DAML+OIL is an ontology language designed specifically for use in the semantic Web. It was produced by merging two ontology languages, OIL and DAML. OIL integrates features from frame-based systems and description logics (DLs), and has an RDF-based syntax. DAML is more tightly integrated with RDF, enriching it with a larger set of ontological primitives [13].

Because DAML+OIL is based on a description logic, a DL reasoner can be used to compare (semantically) descriptions written in DAML+OIL. This provides a powerful framework for defining and comparing e-commerce service descriptions.

Service Description Languages

Choosing the appropriate service ontology is an important part of the matchmaking prototype. The discussion that follows will illustrate why it is reasonable to consider DAML-S in this context.

WSDL

WSDL (Web Services Description Language) is an XML format for describing network services in abstract terms derived from the concrete data formats and protocols used for implementation [28].

As communication protocols and message formats are standardized in the Web community, it becomes possible and important to describe communications in a structured way. WSDL addresses this need by defining an XML grammar for describing network services as collections of communication endpoints capable of exchanging messages. WSDL service definitions provide documentation for distributed systems and serve as a recipe for automating the details involved in application communications.

However, WSDL does not support semantic description of services. For example, it does not support the definition of logical constraints between its input and output parameters, although it has the concept of input and output types as defined by XSD.

UDDI

UDDI (Universal Description, Discovery, and Integration) is another emerging XML-based standard for Web service description [26]. It enables a business to describe its business and services, discover other businesses that offer desired services, and integrate with these other businesses by providing a registry of businesses and Web services.

UDDI describes businesses by their physical attributes, such as name, address, and the services they provide. UDDI descriptions are augmented by a set of attributes, called tModels, that describe additional features, such as the classification of services within taxonomies like NAICS (North American Industry Classification System).

Since UDDI does not represent service capabilities, however, the tModels it uses only provide a tagging mechanism, and the search performed is only done by string matching on some fields they have defined. Thus, it is of no use for locating services based on a semantic specification of their functionality.

DAML-S

DAML-S supplies Web service providers with a core set of markup language constructs for describing the properties and capabilities of their Web services in unambiguous, computer-interpretable form. DAML-S markup of Web services is intended to facilitate the automation of Web service tasks, including automated Web service discovery, execution, interoperation, composition, and execution monitoring [24].

In DAML-S, service descriptions are structured into three essential types of knowledge: a ServiceProfile, a ServiceModel (which describes the ServiceProfile), and a ServiceGrounding. The ServiceProfile is typically required in a matchmaking process because it provides the information needed for an agent to discover a service that meets its requirements.

Paolucci, Kawamura, Payne, and Sycara have described some experiments designed to prove that DAML-S and its ServiceProfile can take up the challenge of representing the functionalities of Web services [20].

Matchmaking Systems

InfoSleuth

InfoSleuth, an agent-based information discovery and retrieval system, adopts "broker agents" to perform the syntactic and semantic matchmaking [18, 19].

The broker agent matches agents that require services with other agents that can provide them. By maintaining a repository containing up-to-date information about the operational agents and their services, the broker enables the querying agent to locate all available agents that provide appropriate services.

Syntactic brokering is the process of matching requests to agents on the basis of the syntax of the incoming messages that wrap the requests. Semantic brokering is the process of matching requests to agents on the basis of the requested agent capabilities or services, with the agent capabilities and services described in a common shared ontology of attributes and constraints. This single domain-specific ontology is a shared vocabulary that all agents can use to specify advertisements and requests to the broker.

In InfoSleuth, service capability information is written in LDL++, a logical deduction language [8]. Agents use a set of LDL++ deductive rules to support inferences about whether an expression of requirements matches a set of advertised capabilities. In contrast, the authors prefer to describe services using a standard ontology language with a declarative semantics. Such descriptions are easy to understand, highly portable, and do not constrain agents to use any particular deductive mechanism.

RETSINA/LARKS

Sycara, Lu, Kilusch, and Widoff and Sycara, Paolucci, van Velsen, and Giampapa have developed a multiagent infrastructure named RETSINA (Reusable Task Structure-based Intelligent Network Agents) [21, 22]. Mediation in this system also relies on service matchmaking, although their specification of capability and service descriptions is different from the one presented here.

They distinguished three general agent categories in cyberspace: service provider, service requester, and middle agent. To describe the capabilities of these agents in the matchmaking process, they have defined and implemented an ACDL (Agent Capability Description Language) called LARKS (Language for Advertisement and Request for Knowledge Sharing). Larks offers the option of using application domain knowledge in any advertisement or request by using a local ontology, written in a specific concept language, ITL, to describe the meaning in a Larks specification.

As with InfoSleuth, the methodology presented here differs from this system in respect to service description language, agent platform, and matching engine. Moreover, the approach adopted here seeks to ensure that the service description language lends itself to the negotiation process, that is, that the same service description language is applicable to the negotiation stage.[2]

DLs and DAML+OIL

The use of description logics and DAML+OIL is central to the approach presented here. Some details of the two formalisms will therefore be helpful in understanding the remainder of the paper.

Description Logics

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 [15]. A DL reasoner can check whether two concepts subsume each other [12].

A DL knowledge base typically consists of two components, a "TBox" and an "ABox." The TBox defines the structure of the knowledge domain and consists of a set of asserted axioms—say, the definition of a new concept in terms of other previously defined concepts. The ABox contains a concrete example of the knowledge domain and asserted axioms about individuals—for example, an individual is an instance of a concept, or an individual is related to another by a role.

In the following sections, DL notations will be used to express our design. Thus, it will be useful to give an overview of DL languages and notations. A detailed discussion of DLs is, however, beyond the scope of this paper, and the interested reader is referred to Baader et al. for further details [1].

Description Logics Syntax

Elementary descriptions are *atomic concepts* and *atomic roles.* Complex descriptions can be built from them inductively with *concept constructors*. The discussion that follows will use abstract notation. The letters A and B are used for atomic concepts, the letter R for atomic roles, and the letters C and D for concept descriptions. Description languages are distinguished by the constructors they provide, and the language $\mathcal{AL}$ is a minimal language that is of practical usage. Concept descriptions in $\mathcal{AL}$ are formed according to the following syntax rule [1]:

$C, D \rightarrow A \mid$ (atomic concept)

$\top$ | (universal concept)
$\bot$ | (bottom concept)
$\neg$ A | (atomic negation)
$C \sqcap D$| (intersection)
$\forall R.C$ | (universal value restriction)
$\exists\ R.\top$ | (limited existential value restriction)

To give examples of what can be expressed in $\mathcal{AL}$, suppose that *Person* and *Female* are atomic concepts. Then *Person* $\sqcap$ *Female* and *Person* $\sqcap \neg$*Female* are $\mathcal{AL}$ concepts describing, intuitively, persons who are female and persons who are not female. If it is supposed, in addition, that *hasChild* is an atomic role, the concepts *Person* $\sqcap\ \exists$*hasChild.*$\top$ and *Person* $\sqcap\ \forall$*hasChild.Female* can be formed, denoting persons who have a child and persons whose children are all female. Using the bottom concept ($\bot$), we can also describe persons without a child by the concept *Person* $\sqcap\forall$*hasChild.*$\bot$ [1].

This basic $\mathcal{AL}$ language does not fulfill the requirements of the present investigation because it is necessary to be able to reason with DAML+OIL descriptions, which include, for example, cardinality restrictions on roles, and data types (integers, strings, etc.). Therefore the DL $\mathcal{SHIQ}$(D) is used, because its expressive power is (almost) equivalent to that of DAML+OIL [11, 13, 15]. This language consists of the basic $\mathcal{AL}$ language plus the negation of arbitrary concepts, (qualified) cardinality restrictions, role hierarchies, inverse roles, transitive roles, and data types (a restricted form of DL concrete domains). A detailed discussion of these and other DL constructors can be found in Baader et al. [1].

The increased expressive power of the language is manifested in a range of additional constructors, including:

$\exists\ R.C$	(full existential value restriction)
$\neg C$	(atomic negation of arbitrary concept)
$\leq n\ R$	(at-most cardinality restriction)
$\geq n\ R$	(at-least cardinality restriction)
$= n\ R$	(exact cardinality restriction)
$\leq n\ R.C$	(qualified at-most cardinality restriction)
$\geq n\ R.C$	(qualified at-least cardinality restriction)
$= n\ R.C$	(qualified exact cardinality restriction)
$\leq_{\mathrm{n}} R$	(concrete domain max restriction)
$\geq_{\mathrm{n}} R$	(concrete domain min restriction)
$=_{\mathrm{n}} R$	(concrete domain exact restriction)

Here are some examples of what can be expressed with these new constructors: If *Woman* $\equiv$ Person $\sqcap$ *Female*, then *Woman* $\sqcap\ \exists$*hasChild.Person* intuitively denotes "mothers," $\neg$*Woman* denotes individuals who are not women, *Mother* $\sqcap\ \geq\ 3$ *hasChild* denotes a mother with more than three children, *Mother*$\sqcap = 3$ *hasChild* denotes a mother with exactly three daughters, and *Person* $\sqcap\ \geq_{18}$ *hasAge* denotes "adults," that is, a person whose age is greater than 18.

DL semantics

In order to define a formal semantics of DLs, it is necessary to consider *interpretations* $\mathfrak{I}$ that consist of a nonempty set $\Delta^{\mathfrak{I}}$ (the domain of the interpretation) and an interpretation function, which assigns to every atomic concept A a set $A^{\mathfrak{I}} \subseteq \Delta^{\mathfrak{I}}$, and to every atomic role R a binary relation $R^{\mathfrak{I}} \subseteq \Delta^{\mathfrak{I}}$. The interpretation of complex concepts is built up from the interpretation of primitive concepts [e.g., $(C \sqcap D)^{\mathfrak{I}} = C^{\mathfrak{I}} \cap D^{\mathfrak{I}}$ and $(\exists R.\top)^{\mathfrak{I}} = \{a \in \Delta^{\mathfrak{I}} \mid \exists b.(a, b) \in R^{\mathfrak{I}}\}$.

We say that two concepts C, D are equivalent, and write $C \equiv D$, if $C^{\mathfrak{I}} = D^{\mathfrak{I}}$ for all interpretations $\mathfrak{I}$. For instance, going back to the semantics of concepts, one can easily verify that $\forall \mathit{hasChild}.\mathit{Female} \sqcap \forall \mathit{hasChild}.\mathit{Student}$ and $\forall \mathit{hasChild}.(\mathit{Female} \sqcap \mathit{Student})$ are equivalent. A complete interpretation function for concept description can be found in Baader et al. [1].

Terminologies

In DLs, a knowledge base (equivalent to an ontology) consists of a set of terminological axioms that asserts how concepts or roles are related to one other. In the most general case, *terminological axioms* have the form:

$$C \sqsubseteq D \ (R \sqsubseteq S) \text{ or } C \equiv D \ (R \equiv S)$$

where C, D are concepts (and R, S are roles). The first kind of axiom is called an *inclusion*, and the second is called an *equivalence.*

An equivalence whose left-hand side is an atomic concept is sometimes called a definition, and can be thought of as introducing *symbolic names* for complex descriptions [1].[3]

DAML+OIL

DAML+OIL is a DL-based Web ontology language. Like any other DL, DAML+OIL describes the structure of a domain in terms of classes (concepts in DL) and properties (roles in DL). DAML+OIL is, in fact, based on the $\mathcal{SHIQ}$ (D) DL, and provides an almost equivalent set of class constructors and class and property axioms (DAML+OIL extends $\mathcal{SHIQ}$(D) with the *oneOf* constructor for defining classes extensionally). Like $\mathcal{SHIQ}$(D), DAML+OIL also supports the use of data types and data values in class description, with DAML+OIL relying on XML Schema data types for this purpose. For a complete description of DAML+OIL, the interested reader is referred to Van Harmelen et al. [27].

The DAML-S Service Ontology

The DAML-S Web service ontology will be used as the basis for representing e-commerce constructs like advertisements and service queries.

DAML-S is a DAML+OIL service-description ontology [24]. Through its tight connection with DAML+OIL, DAML-S supports the need for the semantic representation of services. DAML+OIL allows for subsumption reasoning on concept taxonomies, for the definition of relations between concepts, and for the application of property restrictions on the parameters of service concepts. This means that DAML-S can be used to define the entities in e-commerce life cycles, such as advertisements and requests, and to implement the matchmaking functionalities by using a DL reasoner to compute the subsumption relationships of those concepts.

DAML-S aims to facilitate discovery, execution, interoperation, composition, and execution monitoring of Web services. It defines the notions of a service profile (what the service does), a service model (how the service works), and a service grounding (how to use the service).

A service profile describes who provides the service, the expected quality of the service, and the transformation produced by the service in terms of what it expects to run correctly and what results it produces. Specifically, it specifies the *preconditions* that have to be satisfied for effective use of the service, the *inputs* that the service expects, the *effects* expected from the execution of the service, and the *outputs* returned [24]. Because the behavioral aspects of a service profile are outside the scope of this paper, the discussion here will only concern the fact that a service can be represented by *input* and *output* properties (which represent the functional attributes of a service).

Based on the investigation by Paolucci et al. [20], one may conclude that the ability of DAML-S to describe the semantics of Web services meets the requirements of the matchmaking framework:

- Restrictions and constraints on service descriptions can be expressed.
- It provides the shared semantics needed to achieve interoperability.
- Descriptions are amenable to automated reasoning.
- It provides appropriate support for data types.
- Flexibility is provided by its support for loosely structured descriptions (semistructured data).

Service Description

A Sample Ontology

Since service-description ontologies will have an important role to play in the work discussed here, it was necessary to design a domain-specific sample ontology about the sales of computers to achieve agreement at the semantic level between various parties.[4] The prototype used the OilEd ontology editor to build DAML+OIL ontologies [3]. For the purposes of clarity and compactness, however, this paper will use DL notions in place of the DAML+OIL syntax.

The ontology uses the DAML-S *ServiceProfile* class as a common superclass for the concepts *Advertisement*, *Query*, *Template*, and *Proposal*, so they can be expressed as

$$
\begin{array}{lcl}
\mathit{ServiceProfile} & \sqsubseteq & \top \\
\mathit{Advertisement} & \sqsubseteq & \mathit{ServiceProfile} \\
\mathit{Query} & \sqsubseteq & \mathit{ServiceProfile} \\
\mathit{Template} & \sqsubseteq & \mathit{ServiceProfile} \\
\mathit{Proposal} & \sqsubseteq & \mathit{ServiceProfile}
\end{array}
$$

Two kinds of services are also described in this ontology: *Sales* and *Delivery*. *Sales* describes the sale of an item of *EEquipment* through constraints on the object properties and data type properties, such as the unit price. *Delivery* describes the structure of delivery information by specifying, for example, that there must be exactly one *DeliveryLocation* and exactly one *DeliveryDate*.

In accordance with the DAML-S 0.6 specification, *Sales* also includes the providing and requesting *Actor*s as the values of *providedBy* and *requestedBy* properties. This allows the advertiser and the requester to specify who they are and restrict who they would like to do business with.

$$
\begin{array}{lcl}
\mathit{ServiceProfile} & \sqsubseteq & (=1\ \mathit{providedBy.Actor}) \sqcap \\
 & & (=1\ \mathit{requestedBy.Actor}) \sqcap \\
 & & (=1\ \mathit{item.EEquipment}) \sqcap \\
 & & (=1\ \mathit{hasQuantity.Integer}) \\
 & & (=1\ \mathit{hasUnitPrice.Integer}) \sqcap \\
 & & (=1\ \mathit{canDeliver.Delivery}) \\
\mathit{Delivery} & \sqsubseteq & (=1\ \mathit{location.DeliveryLocation}) \sqcap \\
 & & (=1\ \mathit{date.DeliveryDate}) \\
\mathit{Actor} & \sqsubseteq & (=1\ \mathit{hasName.ActorName}) \sqcap \\
 & & (=1\ \mathit{hasCreditLevel.Integer})
\end{array}
$$

To express the concept computer used in this example, a class PC is defined as a subclass of *EEquipment*, and must have several properties, like *hasProcessor* and *memorySize*.

$$
\begin{array}{lcl}
\mathit{PC} & \sqsubseteq & \mathit{EEquipment} \sqcap \\
 & & (=1\ \mathit{hasProcessor.Processor}) \sqcap \\
 & & (=1\ \mathit{memorySize.positiveInteger}) \\
\mathit{Processor} & \equiv & \mathit{PentiumIII} \sqcup \mathit{Pentium4} \sqcup \mathit{Athlon}
\end{array}
$$

As noted in the preceding section, the service is represented by the input and output properties of the profile. The input property specifies the information that the service requires to proceed with the computation.

For example, a PC-selling service could require such information as unit price and quantity as the inputs to sell. The outputs specify the result of the operation of the service. In the PC-selling case, the output could be an item description that acknowledges the sale.

These restriction properties are divided into *inputs* and *outputs* according to the context in which they are used.[5] In particular, *inputs* are used by buyers and sellers to describe business constraints (e.g., unit quantity, unit price, delivery information), and *outputs* are used to describe the product itself.

$$\begin{array}{lcl} \mathit{inputs} & \sqsubseteq & \mathit{parameter} \\ \mathit{outputs} & \sqsubseteq & \mathit{parameter} \\[1ex] \mathit{hasQuantity} & \sqsubseteq & \mathit{inputs} \\ \mathit{hasUnitPrice} & \sqsubseteq & \mathit{inputs} \\ \mathit{canDeliver} & \sqsubseteq & \mathit{inputs} \\[1ex] \mathit{item} & \sqsubseteq & \mathit{outputs} \end{array}$$

These simple constructs made it possible to express the concepts needed in this context, but arbitrarily complex DAML+OIL constructs could be used if required. The next several sections will show the examples used in the matchmaking process.

Advertisement

Let us now consider the example of an advertisement. Suppose that one wants to specify the concept of an advertisement by which the Actor would like to sell some PCs. There are several restrictions on the *Sales* and the *Delivery*. For example:

- Items are provided by an Actor with name "Georgia."
- Items are PCs and the memory size is at least 128 Mb.
- The quantity of PCs being bought will be less than 200.
- The unit price is more than 700.
- The seller must have a creditLevel greater than 5.
- Goods must be delivered before September 15, 2002.
- Goods must be delivered in Bristol.

In DL notation, this advertisement can be written as:

$$\begin{array}{lcl} \mathit{Advert1} & \equiv & \mathit{ServiceProfile} \sqcap \\ & & (\forall \mathit{providedBy}.(\mathit{Actor} \sqcap \mathit{hasName}.\{\mathit{Georgia}\}) \sqcap \\ & & \forall \mathit{requestedBy}.(\mathit{Actor} \sqcap \geq_{5} \mathit{hasCreditLevel}) \sqcap \\ & & \forall \mathit{item}.(\mathit{PC} \sqcap \geq_{128} \mathit{memorySize}) \sqcap \\ & & \geq_{700} \mathit{hasUnitPrice} \sqcap \\ & & \leq_{200} \mathit{hasQuantity} \sqcap \\ & & \forall \mathit{delivery}.(\mathit{Delivery} \sqcap \\ & & \leq_{20030501} \mathit{date} \sqcap \forall \mathit{location}.\mathit{Manchester})) \end{array}$$

In DAML+OIL, the way to express a concept like "has unit price more than 700" is to define a new data type "more700," and describe it "$\forall$*hasUnitPrice.more700.*" However, for achieving concept reasoning with data types using the RACER reasoner, the RACER syntax is used to express this kind of concept "($\geq_{700}$ *hasUnitPrice*)."

In addition, intuitively, an advertisement should be an instance rather than a concept. That is, it looks more reasonable to express it as $\mathit{Advert1} \in \mathit{ServiceProfile} \sqcap \ldots$. The reason for treating advertisements as TBox concepts is that TBox reasoning is often more effective than ABox reasoning, and is equivalent to

ABox reasoning when the ABox is restricted to assertions of this type. In this example instance advertisements are treated as atomic primitive concepts. Thus, instead of having the ABox assertion a:*C*, a TBox assertion, $C_a \sqsubseteq C$,[6] is used. In contexts like the present e-commerce application, where individuals are not related to one another via properties, this does not lead to any loss of inferential power. Since this issue is beyond the scope of the paper, the interested reader is referred to the treatment by Tessaris [23].

Query

As in the case of the advertisement, a query can be defined by which Actor would like to buy some PCs. For example, restrictions to *Sales* and *Delivery* could express the following:

- The provider is an Actor with creditLevel greater than 5.
- Items are PCs, and the Processor must be Pentium4.
- The unit price must be less than 700.

From the Description Logic point of view, the query and the advertisement are almost identical. Both of them are subsumed by the concept *ServiceProfile.*

Note that the query does not specify anything about the delivery. The flexibility of DL-based languages like DAML+OIL makes it possible to do this and still be able to find relevant matches.

$$\begin{aligned} Query1 \quad \equiv \quad & ServiceProfile \sqcap \\ & (\forall\ providedBy.(Actor \sqcap \geq_5 hasCreditLevel) \sqcap \\ & \forall item.(PC \sqcap 8hasProcessor.Pentium4) \\ & \leq_{700} hasUnitPrice)) \end{aligned}$$

Revised Design

As discussed in the first section, in accordance with DAML-S, the providing and requesting *Actor*s have been included as the values of *providedBy* and *requestedBy* properties in the definitions of advertisements and queries. This design looks reasonable and rational but has a fatal error.

Consider the advertisement *Advert1*, which has the property *providedBy*.(*Actor* ⊓ ∀*hasName*.{*Georgia*}). Consider the request *Query1*, and suppose that we perform a matchmaking operation between *Advert1* and *Query1* using a DL reasoner to (semantically) compare the DAML+OIL descriptions. Owing to the existence of *providedBy*.(*Actor* ⊓ ∀*hasName*.{*Georgia*}), there is no subsumption relationship between *Query1* and *Advert1*. All one can do is prove that the two descriptions are not incompatible (their intersection is not equivalent to the bottom concept). This is probably the case because the requester could not have the knowledge that the looked-for service will be provided by an *Actor* with the name "Georgia." Matches of this kind are weak and do not allow for result selection via a hierarchy of match types with varying specificity (discussed below).

This design problem is inherent to the DAML-S specification. There is too much information inside the service profile, and this makes it difficult to use automated reasoning techniques to compute semantic matches between service descriptions.

The problem is fixed by modifying the design of advertisements and queries. The new design treats advertisements and queries as objects with various properties, one of which is the profile. Information about who is providing and requesting services is removed from the service profile and attached to advertisements and queries via the *providedBy* and *requestedBy* properties (this could be thought of as some extra information provided by the advertiser/ querier). The core *ServiceProfile* component is attached to advertisements and queries via the *profile* property, and includes constraints such as item information, unit price, unit quantity, and delivery information. Later, in the matchmaking phase, only this *ServiceProfile* part of an advertisement will be used when computing semantic matches. Constraints such as *hasCreditLevel* might also be used in realistic e-commerce applications, such as eBay (www.ebay.com) or Amazon (www.amazon.com), but we do not consider them in the prototype.

The modified design uses the following notation to separate the different components of an advertisement:

$$\begin{aligned} \mathit{Advert1} \quad = \quad & (\mathit{providedBy}\ (\mathit{Actor} \sqcap \forall \mathit{hasName}.\{\mathit{Georgia}\}),\\ & \mathit{requestedBy}\ (\mathit{Actor} \sqcap \geq_5 \mathit{hasCreditLevel}),\\ & \mathit{profile}\ (\mathit{ServiceProfile} \sqcap\\ & \forall \mathit{item}.(\mathit{PC} \sqcap \geq_{128} \mathit{memorySize}) \sqcap\\ & \geq_{700} \mathit{hasUnitPrice} \sqcap\\ & \leq_{200} \mathit{hasQuantity} \sqcap\\ & \forall \mathit{canDeliver}.(\mathit{Delivery} \sqcap\\ & \leq_{20030501} \mathit{date} \sqcap \forall \mathit{location}.\mathit{Manchester})) \end{aligned}$$

Similarly, queries are written as:

$$\begin{aligned} \mathit{Query1} \quad = \quad & (\mathit{providedBy}\ (\mathit{Actor} \sqcap \geq_5 \mathit{hasCreditLevel}),\\ & \mathit{profile}\ (\mathit{ServiceProfile} \sqcap\\ & \forall \mathit{item}.(\mathit{PC} \sqcap \forall \mathit{hasProcessor}.\mathit{Pentium4}) \sqcap\\ & \leq_{700} \mathit{hasUnitPrice})) \end{aligned}$$

Matchmaking Operation

Matching Definition

Matchmaking is defined as a process that requires a repository host to take a query or an advertisement as input, and to return all the advertisements that may satisfy the requirements specified in the input query or advertisement.[7] 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 that are compatible, *matches*(*Q*):

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

Two descriptions are compatible if their intersection is satisfiable:

$satisfiable(D1, D2) \Leftrightarrow \neg(D1 \sqcap \mathrm{D2} \sqsubseteq \top)$

For example, consider the following query:

$Query2 = (providedBy\ (Actor \sqcap \forall hasName.\{Alan\}),$
$requestedBy\ (Actor\ \sqcap =_5 hasCreditLevel),$
$profile\ (ServiceProfile\ \sqcap$
$\forall item.(PC\ \sqcap =_{256} memorySize)\ \sqcap$
$=_{500} hasUnitPrice))$

The intersection of this query with *Advert1* is satisfiable. Formally,

$Advert1 \in matches(Query2)$

Matching Algorithm

To understand the matching algorithm adopted in the prototype, it is first necessary to introduce the definition of the degree of match. This notion is introduced because it is not particularly useful merely to determine that an advertisement and query are not semantically incompatible. Therefore, starting from the matching degree definition described by Paolucci et al. [20], the match level "intersection satisfiable" is extended to:

- **Exact:** If advertisement *A* and request *R* are equivalent concepts, then one calls the match Exact; formally, $A \equiv R$.
- **PlugIn:** If request *R* is a subconcept of advertisement *A*, then one calls the match PlugIn; formally, $R \sqsubseteq A$.
- **Subsume:** If request *R* is a superconcept of advertisement A, then one calls the match Subsume; formally, $A \sqsubseteq R$.
- **Intersection:** If the intersection of advertisement *A* and request *R* is satisfiable, then one calls the match Intersection to distinguish it from Disjoint, where the advertisement and request are completely incompatible (this distinction was not made by Paolucci et al. [20]); formally, $\neg(A \sqcap R \sqsubseteq \bot)$.
- **Disjoint:** Otherwise, one calls the match Disjoint; that is, $A \sqcap R \sqsubseteq \bot$.

Degrees of the match are organized on a discrete scale. Exact matches are clearly preferable. PlugIn matches are considered the next-best, because advertisers may be expected to also provide specific (subclass) services. For example, an advertiser selling PCs might be expected to sell specific kinds of PCs. Subsume matches are considered to be third-best, because an advertiser may also provide specific (superclass) services. For example, an advertiser selling used PCs may sell PCs in general.[8] Intersection is considered to be

```
doMatch(Request) {
    forall advertisements in Repository do {
        globalDegreeMatch = Exact
        degreeMatch = matchDegree(RqInputs, AdInputs)
        if (degreeMatch < globalDegreeMatch)
            globalDegreeMatch = degreeMatch
        degreeMatch = matchDegree(RqOutputs, AdOutputs)
        if (degreeMatch < globalDegreeMatch)
            globalDegreeMatch = degreeMatch
        storeResult(currentAdvertisement, globalDegreeMatch)
    }
}
matchDegree(R, A) {
    if concept-equivalent(R, A) return Exact
    if concept-subsumes(A, R) return PlugIn
    if concept-subsumes(R, A) return Subsume
    if concept-subsumes(:R, A) return Disjoint
    return Intersection
}
```

Figure 1. Pseudocode for Request Matching

fourth-best. It only says that the advertisement is not incompatible with the request. Disjoint is the lowest level, because it shows that no item could satisfy both the advertisement and the request—it is considered to be a failed match.

With these definitions of match degrees, the process of matching a request can now be introduced. The RACER system is used to compute a ServiceProfile hierarchy for all advertised services. For an incoming request, RACER is used to classify the input/output parts of the request's ServiceProfile *R* (i.e., to compute the input/output parts of *R*'s subsumption relationships with respect to the input/output parts of all the advertisement ServiceProfiles). To express it precisely, a piece of pseudocode is presented in Figure 1. In the pseudocode, the inequations like "degreeMatch < globalDegreeMatch" follow the definition of match ordering (i.e., "Disjoint < Intersection < Subsume < PlugIn < Exact").

Prototype Implementation

This section describes the implementation of a multiagent system that includes matchmaking, advertising, and querying agents. The system emulates a simple but realistic e-commerce scenario. Some issues, however, such as security (e.g., fraud), have not been taken into account, because they were not considered relevant to the purpose of investigating ontology-based service description and a DL-based matchmaking service.

Abstract Roles

The usability of service descriptions and matchmaking in the semantic Web will be tested by introducing a scenario in which agents play a variety of roles.

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

All three of these abstract roles can 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, different types of matchmaking systems can be covered by adding one or more roles to the concrete entity in the real system.

Functionalities

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

Advertising

The Advertiser publishes to the Host a service description of what it is providing or seeking. This description captures the relevant features of the service, including the service profile component that will be used in matchmaking.

Querying

The Seeker can submit a query to find relevant advertisements among the currently available ones. By adding constraints on aspects that the Seeker is interested in, the query can be used to filter irrelevant advertisements. Two kinds of queries can be defined:

- *Volatile Query:* The seeker submits a query to the Host, the matched advertisements are immediately returned, and then the Host discards the query.
- *Persistent Query:* The seeker can also submit a persistent query to the Host. A persistent query is a query that will remain valid for a length of time defined by the Seeker. The Host immediately returns matched advertisements that are currently present in the repository. Within the validity period of the query, whenever a matching advertisement is added to the repository (or an advertisement is modified so that it becomes a match), the Host will notify the Seeker with a new set of matched advertisements including those that have been changed or added. The persistent query is automatically removed when the validity period is ended.

Modifying/Withdrawing

An Advertiser can modify and withdraw the advertisements it has published before. After the Advertiser publishes an advertisement to the Host, the Host notifies an ID indicating the advertisement to the Advertiser. Later on, this ID is used between the Host and the Advertiser to specify which advertisement is to be modified or withdrawn. There is an obvious security issue involved, but it is assumed that all the partners in the framework are trusted.

Browsing

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

Agents

JADE was chosen as the agent platform because the goal of JADE is to simplify the development of multiagent 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 one 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 the responsibility to initialize the RACER server using assigned ontologies and maintain the advertisement repository. This is the core component of the system, and its operation is described in more detail below.
- *AdvertiserAgent* publishes the advertisement to the HostAgent, withdraws, and modifies its own advertisement if needed. It can also browse the advertisement repository in HostAgent.
- *SeekerAgent* has the choice of publishing a volatile or persistent request to the HostAgent. It also has the browse functionality.

Matchmaking

At the beginning of the matchmaking process, the HostAgent initializes RACER with the service ontology described. The RACER system uses the ontology to compute the subsumption relations between advertisements and requests throughout the whole matchmaking process. On receiving an advertisement, the HostAgent assigns it a unique ID and stores it in the repository. It then sends the advertisement's ServiceDescription to the RACER system to be added to the subsumption hierarchy.

On receiving a request, the HostAgent uses the RACER system to compute all the match degrees between the request and each advertisement in the repository, as described. Matching advertisements are returned to the seeker agent, along with their IDs and match degrees (Exact, PlugIn, etc.). For efficiency reasons, match results for a persistent request are maintained until the request expires.

The HostAgent stores persistent requests along with an ID and expiry duration. At the same time that it classifies new (and updated) advertisements, the HostAgent will check all persistent requests, delete them if expired, and compute their match degree with respect to the new (or updated) advertisement. If a match is found, the information is added to the stored information from the initial matchmaking, and the complete result for the persistent request is returned to the seeker agent.

Evaluation

In terms of functionality, the matchmaking stage has achieved its purpose: It can respond to an input request with the results of matched advertisements. However, to find a match for a particular request, the RACER reasoner needs to check the satisfiability of the request with each advertisement already published to the Matchmaking host. Given the high worst-case complexity of reasoning with DAML+OIL descriptions,[9] the question of scalability arises. Therefore the prototype implementation was used to carry out some simple experiments designed to test the system's performance in a realistic agent-based e-commerce scenario. The experiment used datasets of between 100 and 1,500 advertisements, and recorded the time the DL reasoner needed to find matched advertisements in response to a given request. The data sets were artificially generated by randomly creating the specifications of advertisements. For example, the range of *location* and *hasProcessor* were randomly chosen from a set of concepts in the ontology, and the numbers of *memorySize* and *hasUnitPrice* were randomly chosen from a fixed range of integers. All the experiments were performed on a machine equipped with a Pentium III 850 MHz processor, with 256 MB of main memory and running LINUX with the kernel version 2.4.9.

The results showed that, regardless of the number of advertisements, if the advertisements have already been classified (in RACER's TBox), then the reasoning time required to respond to a matching request is always less than 20 milliseconds—so small that accurate measurement was difficult. This would be fast enough for the matchmaking system to handle a high frequency of matching requests.

In contrast, classifying the advertisements in the TBox is time-consuming. From the comparison of the different-size data sets shown in Figure 2, one can see that the average classification time per advertisement (shown on the *y*-axis) increases rapidly with the size of the data set (shown on the *x*-axis). The time rises from 49.57 milliseconds per advertisement for a data set of size 100, and increases to 715.33 milliseconds per advertisement for a data set of size 1,500.

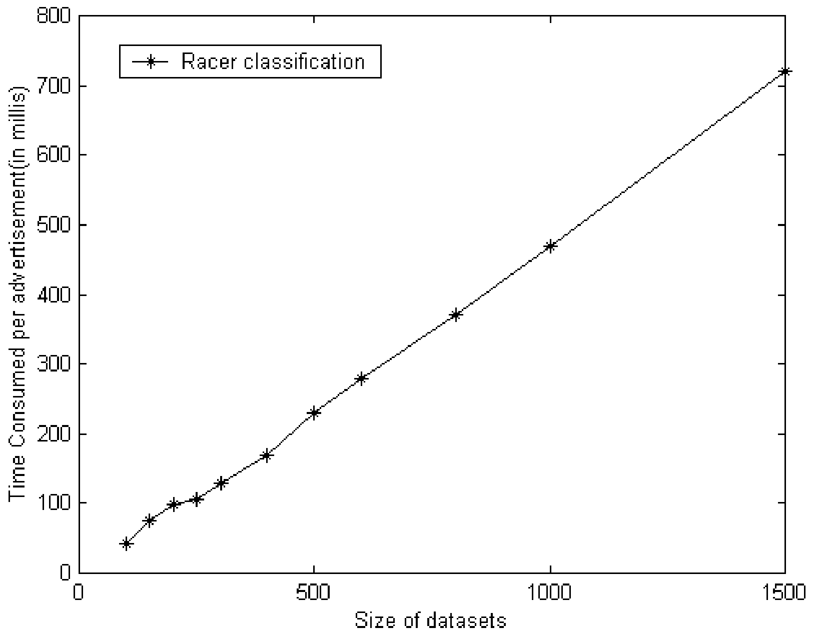

Figure 2. RACER Classification Times

Although this test illustrates that data-set size is an important issue in applications that use a DL reasoner, it does not mean that large data sets cannot be handled. For instance, in the prototype, the TBox classification could be done off-line. That is, for all the published advertisements, the TBox is classified before the matchmaking process starts, and the classified TBox is used to reason about requests. As for new incoming advertisements, they can simply be inserted into the classified TBox hierarchy, which is *much* easier than classifying the entire TBox.[10]

Conclusion

This paper introduces service matchmaking in e-commerce, assesses the requirements for a service-description language and ontology, and argues that DAML+OIL and DAML-S fulfill these requirements. The argument is supported by the design and implementation of a prototype matchmaker that uses a DL reasoner 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 behavior of an intelligent agent to approach more closely that of a human user trying to locate suitable Web services (assuming that a suitable ontology has already been developed and deployed).

The design of the prototype matchmaker revealed a problem with the use of DAML-S in matchmaking: DAML-S service profiles contain too much information for effective matching. This problem was solved by separating various components of the description. In particular, the description of the service being provided was separated from the descriptions of the providing and requesting "actors."

Finally, the performance of the prototype implementation was evaluated using a simple but realistic e-commerce scenario. This revealed that, although the initial classification of large numbers of advertisements could be quite time-consuming, subsequent matching of queries to advertisements could be performed efficiently. Based on these preliminary results, it seems possible that DL reasoning technology can cope with large-scale e-commerce applications. Future work will include more extensive testing to establish whether this is the case.

NOTES

1. Coincidentally, a similar approach has been adopted by Di Noia, Di Sciascio, Donini, and Mongiello [9].
2. After finding suitable services, a consuming agent may enter into a negotiation with the providing agent regarding the terms of service provision (cost, delivery, etc.)
3. This does not hold in the general case where the knowledge base can contain arbitrary axioms.
4. Note that this simple ontology is only intended for didactic purposes. In realistic applications, much larger and more comprehensive ontologies would be required.
5. The matchmaking algorithm is based on this division.
6. The idea is to create a "pseudo-concept," C_a, which is a subconcept of C. Thus, an individual instantiation is expressed using a concept implication.
7. It is obvious that the host needs to return advertisements on receiving a query, but it is also reasonable for the host to return advertisements on receiving an advertisement; for example, an advertiser might want to know the advertisements made by the others so that he can make some modification to his business strategy.
8. One could argue that Subsume is preferable to PlugIn, but this discussion is beyond the scope of the present work and would not qualitatively affect the performance of the prototype.
9. Key inference problems for the logic implemented in the RACER system have worst-case EXPTIME complexity in the size of the input.
10. However, removing advertisements from the TBox hierarchy would be more difficult.

REFERENCES

1. Baader, F.; Calvanese, D.; McGuinness, D.; Nardi, D.; and Patel-Schneider, P.F., eds. *The Description Logic Handbook: Theory, Implementation and Applications.* Cambridge: Cambridge University Press, 2003.
2. Bartolini, C.; Preist, C.; and Jennings, N. Architecting for reuse: A software framework for automated negotiation. In F. Giunchiglia, J. Odell, and G. Weiss (eds.), *Proceedings of the Third International Workshop on Agent-Oriented Software Engineering*. Bologna: Springer, 2002, pp. 88–100.
3. Bechhofer, S.; Horrocks, I.; Goble, C.; and Stevens, R. OilEd: A reasonable ontology editor for the semantic Web. In *Proceedings of the Joint German/*

Austrian Conference on Artificial Intelligence (KI 2001), no. 2174 in Lecture Notes in Artificial Intelligence. Vienna: Springer, 2001, pp. 396–408.

4. Bechhofer, S.; van Harmelen, F.; Hendler, J.; Horrocks, I.; McGuinness, D.L.; Patel-Schneider, P.F.; and Stein, L.A. OWL Web ontology language 1.0 reference. W3C proposed recommendation (www.w3.org/TR/owl-ref).

5. Bellifemine, F.; Poggi, A.; and Rimassa, G. JADE—A FIPA2000-compliant agent development environment. In J. Müller, E. Andre, S. Sen, and C. Frasson (ed.), *Proceedings of the Fifth International Conference on Autonomous Agents*. Montreal: ACM, 2001, pp. 216–217.

6. Berners-Lee, T. *Weaving the Web.* San Francisco: Harper, 1999.

7. Brickley, D., and Guha, R.V. RDF vocabulary description language 1.0: RDF schema. W3C proposed recommendation (www.w3.org/TR/rdf-schema).

8. Chimenti, D.; Gamboa, R.; Krishnamurthy, R.; Naqvi, S.A.; Tsur, S.; and Zaniolo, C. The LDL system prototype. *IEEE Transactions on Knowledge and Data Engineering, 2,* 1 (1990), 76–90.

9. Di Noia, T.; Di Sciascio, E.; Donini, F.M.; and Mongiello, M. A system for principled matchmaking in an electronic marketplace. *International Journal of Electronic Commerce, 8,* 4 (summer 2004), 9–37.

10. Haarslev, V., and Moller, R. RACER system description. In R. Goré, A. Leitsch, and T. Nipkow (ed.), *Proceedings of the International Joint Conference on Automated Reasoning (IJCAR 2001).* Siena: Springer, 2001, pp. 701–705.

11. Horrocks, I. DAML+OIL: A reason-able Web ontology language. In C.S. Jensen, K.G. Jeffery, J. Pokorny, S. Saltenis, E. Bertino, K. Bohm, and M. Jarke (eds.), *Proceedings of EDBT 2002.* Prague: Springer, March 2002, pp. 2–13.

12. Horrocks, I., and Patel-Schneider, P.F. Comparing subsumption optimizations. In E. Franconi, G. De Giacomo, R.M. MacGregor, W. Nutt, and C.A. Welty (eds.), *Proceedings of the 1998 Description Logic Workshop (DL'98).* Trento, Italy: CEUR Electronic Workshop Proceedings, 1998, pp. 90–94 (http://ceur-ws.org/Vol-11/).

13. Horrocks, I.; Patel-Schneider, P.F.; and van Harmelen, F. Reviewing the design of DAML+OIL: An ontology language for the semantic Web. In R. Dechter, M. Kearns, and R. Sutton (ed.), *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI 2002)*. Edmonton: AAAI Press, 2002, pp. 792–797.

14. Horrocks, I., and Sattler, U. Ontology reasoning in the SHIQ(D) description logic. In B. Nebel (ed.), *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001)*. San Francisco: Morgan Kaufmann, 2001, pp. 199–204.

15. Horrocks, I.; Sattler, U.; and Tobies, S. Practical reasoning for expressive description logics. In H. Ganzinger, D. McAllester, and A. Voronkov (eds.), *Proceedings of the Sixth International Conference on Logic for Programming and Automated Reasoning (LPAR'99).* Lecture Notes in Artificial Intelligence, No. 1705. Tbilisi: Springer, 1999, pp. 161–180.

16. Horrocks, I.; Sattler, U.; and Tobies, S. Reasoning with individuals for the description logic SHIQ(D). In D. McAllester (ed.), *Proceedings of the 17th International Conference on Automated Deduction (CADE2000)*. Pittsburgh: Springer, 2000, pp. 482–496.

17. McGuinness, D.L. Ontological issues for knowledge-enhanced search. In

N. Guarino (ed.), *Proceedings of the First International Conference on Formal Ontology in Information Systems.* Trento, Italy: IOS Press, 1998, pp. 302–316.
18. Nodine, M.; Bohrer, W.; and Ngu, A. Semantic multibrokering over dynamic heterogeneous data sources in InfoSleuth. In M. Kitsuregawa, L. Maclaszek, M. Papazoglou, and C. Pu (ed.), *Proceedings of the Fifteenth International Conference on Data Engineering*. Sydney: IEEE Computer Society Press, 1999, pp. 358–365.
19. Nodine, M.H.; Fowler, J.; Ksiezyk, J.; Perry, B.; Taylor, M.; and Unruh, A. Active information gathering in InfoSleuth. *International Journal of Cooperative Information Systems, 9,* 1/2 (2000), 3–28.
20. Paolucci, M.; Kawamura, T.; Payne, T.; and Sycara, K. Semantic matching of Web services capabilities. In I. Horrocks and J. Hendler (eds.), *Proceedings of the First International Semantic Web Conference (ISWC).* Sardinia: Springer, 2002, pp. 333–347.
21. Sycara, K.; Lu, J.; Klusch, M.; and Widoff, S. Dynamic service matchmaking among agents in open information environments. *ACM SIGMOD Record* (Special Issue on Semantic Interoperability in Global Information Systems)*, 28,* 1 (1999), 47–53.
22. Sycara, K.; Paolucci, M.; van Velsen, M.; and Giampapa, J. The RETSINA MAS infrastructure. Technical report CMU-RI-TR-01–05. Pittsburgh: Carnegie Mellon University, Robotics Institute, 2001.
23. Tessaris, S. Questions and answers: Reasoning and querying in description logic. Ph.D. dissertation, University of Manchester, 2001.
24. The DAML Services Coalition. DAML-S: Semantic markup for Web services (www.daml.org/services/daml-s/0.9/daml-s.html).
25. Trastour, D.; Bartolini, C.; and Preist, C. Semantic Web support for the business-to-business e-commerce lifecycle. In D. Lassnor, D. De Rourre, and A. Iyengar (ed.), *Proceedings of the Eleventh International Conference on World Wide Web*. New York: ACM, 2002, pp. 89–98.
26. UDDI. White paper (www.uddi.org).
27. van Harmelen, F.; Patel-Schneider, P.F.; and Horrocks, I. Reference description of the DAML+OIL (March 2001) ontology markup language (www.daml.org/2001/03/reference.html).
28. Web services description language (WSDL) 1.1. W3C note, March 15, 2001 (www.w3.org/TR/wsdl).

LEI LI (lil@cs.man.ac.uk) is a Ph.D. student working with Professor Ian Horrocks in the computer science department at the University of Manchester. His primary research interest is the application of description logic reasoning, especially the deployment of ontologies in the semantic Web. He is particularly interested in reasoning with large numbers of individuals using a combination of databases and DL reasoning techniques. He has a bachelor's degree in computer science from the University of Science and Technology of China, and a master's degree in advanced computer science from the University of Manchester. Prior to studying in Manchester, he worked in industry for two years as a software developer in Shanghai, and from April to September 2002, he was a research intern in the Intelligent Enterprise Technology Lab, Hewlett-Packard Labs, Bristol, UK.

IAN HORROCKS (horrocks@cs.man.ac.uk) is professor of computer science at the University of Manchester. His FaCT system revolutionized the design of description

logic systems, redefining the notion of tractability and establishing a new standard for implementations. He has published widely in leading journals and conferences, winning the best paper prize at Knowledge Representation '98. He has been a member of the program/editorial committees of numerous international conferences, workshops, and journals, and was the program chair of the 2002 International Semantic Web Conference, and the semantic Web track chair for the 2003 World Wide Web Conference. He is a member of the Joint EU/US Committee on Agent Markup Languages and the W3C Web Ontology Language working group, and has been involved in the development of the OIL, DAML+OIL, and OWL ontology languages.