A Software Framework For Matchmaking Based on Semantic Web Technology

Lei Li and Ian Horrocks
Department of Computer Science
University of Manchester
Oxford Road, Manchester M13 9PL, 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, standardisation 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 describe the design and implementation of a service matchmaking prototype which 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 behaviour of an intelligent agent to approach more closely that of a human user trying to locate suitable web services. We also present the results of initial experiments testing the performance of this prototype implementation in a realistic agent based e-commerce scenario.

Keywords

Semantic Web, Web Services, Ontologies, Matchmaking

1 Introduction

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 [6].

With a Semantic Web, it is widely believed that 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 [16].

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.) [25].

As a first step in realising the Semantic Web, new standards for defining and using ontologies are already being developed. RDF, which is being developed by the W3C RDF Core working group, is a web markup language that provides basic ontological primitives [7]; DAML+OIL is an ontology language that extends RDF with a much richer set of primitives (e.g., boolean operators and cardinality constraints), and which is now the basis for the W3C Web Ontology Language working group's development of the OWL ontology language standard [10, 4].

Moreover, if applications are to exchange semantic information, they will need to use common ontologies. One such ontology (written in DAML+OIL), which has been designed for the purpose of describing web services, is the DAML-S ontology [24]. In this paper, we present 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, i.e., a repository where agents can advertise and search for services that match some semantic description. We used JADE [5] as the agent platform for our prototype and we used the RACER DL reasoner [9] in order to compute semantic matches between service advertisements and service requests. We illustrate some difficulties both in the application of the DAML-S ontology and in the use of the DL reasoner, and show how these were overcome in the prototype implementation. Finally, we carry out a performance analysis using the prototype in order to discover if the approach is likely to be feasible in large scale web applications.

2 Background

In this section we will give an overview of Semantic Web languages and technologies that are relevant to the prototype implementation.

2.1 Ontology Languages

As we have already mentioned, ontologies play a key role in the Semantic Web by providing vocabularies that can be used by applications in order to understand shared information.

DAML+OIL is an ontology language that has been designed specifically to be used in the Semantic Web. DAML+OIL is the result of merging two ontologies languages: OIL and DAML. OIL integrates the 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 [12].

Because DAML+OIL is based on a Description Logic, we can use a DL reasoner to compare (semantically) descriptions written in DAML+OIL. We believe that this provides a powerful framework for defining and comparing e-commerce service descriptions. In Section 3 we will discuss in more detail DLs in general and DAML+OIL in particular.

[1]Coincidentally, a similar approach has been adopted by Di Noia et al, the interested reader is referred to [19].

2.2 Service Description Languages

In this section, we will discuss an important part of the matchmaking prototype — choosing the appropriate service ontology, and we will illustrate why it is reasonable to consider DAML-S in this context.

2.2.1 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 increasingly possible and important to be able to describe the communications in some 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.

2.2.2 UDDI

Another emerging XML based standard for web service description is UDDI (Universal Description, Discovery and Integration) [26]. It enables a business to (i) describe its business and its services, (ii) discover other businesses that offer desired services, and (iii) 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 that they provide. In addition, UDDI descriptions are augmented by a set of attributes, called tModels, which describe additional features such as the classification of services within taxonomies such as NAICS (North American Industry Classification System).

However, because UDDI does not represent service capabilities, the tModels they use 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 on the basis of a semantic specification of their functionality.

2.2.3 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. In a matchmaking process, the ServiceProfile is typically required, since it provides the information needed for an agent to discover a service that meets its requirements.

In [20], Paolucci et al. have described some experiments designed to prove that DAML-S and its Service Profile can take up the challenge of representing the functionalities of web services.

2.3 Matchmaking Systems

In this section, we will briefly examine other approaches to the matchmaking problem.

2.3.1 InfoSleuth

InfoSleuth [18, 17], an agent-based information discovery and retrieval system, adopts “broker agents” to perform the syntactic and semantic matchmaking.

The broker agent matches agents that require services with other agents that can provide those services. 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 which 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 being 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, the service capability information is written in LDL++ [8], a logical deduction language. 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, we prefer to describe services using a standard ontology language with a declarative semantics. Such descriptions are easy to understand, are highly protable and do not contrain agents to use any particular deductive mechanism.

2.3.2 RETSINA/LARK

Sycara et al. 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 ours.

They distinguished three general agent categories in Cyberspace: service provider, service requester, and middle agent. To describe these agents' capabilities 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 to use 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, our methodology differs from this system in the aspects of service description language, agent platform and matching engine. Moreover, in our approach we want to be sure that the service description language also lends itself to the negotiation process,[2] i.e., the same service description language should be applicable to the negotiation stage.

3 DL and DAML+OIL

The use of Description Logics and DAML+OIL are central to our approach. Some details of the two formalisms will therefore be helpful in understanding the remainder of the paper.

[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.) [2].

3.1 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 [14]. A DL reasoner can check whether two concepts subsume each other [11].

A DL knowledge base typically consists of two components— “TBox” and “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. ABox contains a concrete example of the knowledge domain and asserted axioms about individuals, e.g., an individual is an instance of a concept; or an individual is related to another by a role.

In the following sections, we will use DL notations to express our design. Hence 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 [1] for further details.

3.1.1 Description logics syntax

Elementary descriptions are *atomic concepts* and *atomic roles*. Complex descriptions can be built from them inductively with *concept constructors*. In the following, we will use abstract notation. We use the letters **A** and **B** 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]:

$$\begin{array}{lll} C, D & \rightarrow & A \mid \ (atomic\,concept) \\ & & \top \mid \ (universal\,concept) \\ & & \bot \mid \ (bottom\,concept) \\ & & \neg A \mid \ (atomic\,negation) \\ & & C \sqcap D \mid \ (intersection) \\ & & \forall R.C \mid \ (value\,restriction) \\ & & \exists R.\top \ (limited\,existential\,quantification) \end{array}$$

To give examples of what can be expressed in $\mathcal{AL}$, we suppose that `Person` and `Female` are atomic concepts. Then $Person \sqcap Female$ and $Person \sqcap$

$\neg Female$ are $\mathcal{AL}$-concepts describing, intuitively, those persons that are female, and those that are not female. If, in addition, we suppose that `hasChild` is an atomic role, we can form the concepts $Person \sqcap \exists hasChild.\top$ and $Person \sqcap \forall hasChild.Female$, denoting those persons that have a child, and those persons all of whose children are female. Using the bottom concept ($\bot$), we can also describe those persons without a child by the concept $Person \sqcap \forall hasChild.\bot$ [1].

This basic $\mathcal{AL}$-language does not fulfil the requirements of our investigation as we need to be able to reason with DAML+OIL descriptions, which include, e.g., cardinality restrictions on roles, and datatypes (integers, strings, etc.). We therefore use the DL $\mathcal{SHIQ}(\mathbf{D})$, whose expressive power is (almost) equivalent to that of DAML+OIL [15, 13, 10]. 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 datatypes (a restricted form of DL concrete domains). A detailed discussion of these and other DL constructors can be found in [1].

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

$\exists R.C$	$(full\ existential\ quantification)$
$\neg C$	$(negation\ of\ arbitrary\ concepts)$
$\leq n\ R$	$(atmost\ cardinality\ restriction)$
$\geq n\ R$	$(atleast\ cardinality\ restriction)$
$= n\ R$	$(exactly\ cardinality\ restriction)$
$\leq n\ R.C$	$(qualified\ atmost\ cardinality\ restriction)$
$\geq n\ R.C$	$(qualified\ atleast\ cardinality\ restriction)$
$= n\ R.C$	$(qualified\ exactly\ cardinality\ restriction)$
$\leq_n\ R$	$(concrete\ domain\ max\ restriction)$
$\geq_n\ R$	$(concrete\ domain\ min\ restriction)$
$=_n\ R$	$(concrete\ domain\ exactly\ restriction)$

As 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 that are not women; $Mother \sqcap \geq 3\, hasChild$ denotes a mother with more than three children; $Mother \sqcap = 3\, hasChild.female$ denotes a mother with exactly three daughters; $Person \sqcap \geq_{18} hasAge$ denotes "adults", i.e., a person whose age is greater than 18.

3.1.2 DL Semantics

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

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

3.1.3 Terminologies

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

$$C \sqsubseteq D\ (R \sqsubseteq S) \quad or \quad C \equiv D\ (R \equiv S)$$

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

3.2 DAML+OIL

DAML+OIL is a DL based Web ontology language. As with 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 $\mathcal{SHIQ}(\mathbf{D})$ DL, and provides an almost equivalent set of class constructors and class and property axioms (DAML+OIL extends $\mathcal{SHIQ}(\mathbf{D})$ with the `oneOf` constructor

[3]This does not really hold up in the general case where the knowledge base can contain arbitrary axioms.

for defining classes extensionally). Like $\mathcal{SHIQ}(\mathbf{D})$, DAML+OIL also supports the use of datatypes and data values in class description, with DAML+OIL relying on XML Schema datatypes for this purpose. For a complete description of DAML+OIL, the interested reader is referred to [27].

4 The DAML-S Service Ontology

We have chosen to use the DAML-S web service ontology as the basis to represent e-commerce constructs like advertisements and service queries.

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

DAML-S aims at facilitating 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 in order to run correctly and what results it produces. Specifically, it specifies the *preconditions* that have to be satisfied to use the service effectively; the *inputs* that the service expects; the expected *effects* that result from the execution of the service and the *outputs* returned [24]. Since the behavioural aspects of a service profile are outside the scope of this work, we will only interest ourselves in the fact that a service can be represented by *inputs* and *outputs* properties (which represent the functional attributes of a service).

Through the investigation in [20], we have concluded that the ability of DAML-S to describe the semantics of web services meets the requirements of our matchmaking framework:

- Restrictions and constraints on service descriptions could be expressed;
- It provides the shared semantics needed to achieve interoperability;

- Descriptions are amenable to automated reasoning;
- It provides appropriate support for datatypes;
- Flexibility is provided by support for loosely structured descriptions (semi-structured data).

5 Service Description

In this section, we explain how we use DAML+OIL and the DAML-S ontology to capture the various descriptions that are used in the e-commerce life-cycle.

5.1 A Sample 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 in order to achieve agreement at the semantic level between various parties.[4] In our prototype, we used the OilEd [3] ontology editor to build DAML+OIL ontologies. For the purpose of clarity and compactness, however, in this paper we will use the DL notions in place of the DAML+OIL syntax.

In our ontology, we use the DAML-S `ServiceProfile` class as a common superclass for concept `Advertisement`, `Query`, `Template` and `Proposal`, so they can be expressed as:

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

We also defined two kinds of services in this ontology: `Sales` and `Delivery`. `Sales` describes the sale of an item of `EEquipment` through constraints on the object properties and datatype properties such as the unit price. `Delivery` describes the structure of delivery information by specifying, e.g., that there must be exactly one `DeliveryLocation` and exactly one `DeliveryDate`.

[4]Note that this simple ontology is only intended for didactic purposes. In realistic applications a much larger and more comprehensive ontology would of course be required.

In accordance with the DAML-S 0.6 specification, in `Sales`, we also include the providing and requesting `Actor`s as the values of `providedBy` and `requestedBy` properties. By doing so, we allow the advertiser and the requester to specify who they are and restrict who they would like to do business with.

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

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

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

As noted in Section 4, the service is represented by input and output properties of the profile. The input property specifies the information that the service requires to proceed with the computation.

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

In our work, we divide these restriction properties into *inputs* and *outputs* according to the context in which they are used.[5] In particular, *inputs* are used by

[5]Our matchmaking algorithm is based on this division.

buyers and sellers to describe business constraints (e.g., unit quantity, unit price and delivery information), while *outputs* are used to describe the product itself.

$$\begin{aligned} inputs &\sqsubseteq parameter \\ outputs &\sqsubseteq parameter \\ \\ hasQuantity &\sqsubseteq inputs \\ hasUnitPrice &\sqsubseteq inputs \\ canDeliver &\sqsubseteq inputs \\ \\ item &\sqsubseteq outputs \end{aligned}$$

These simple constructs allowed us to express the concepts we needed in this context, but arbitrarily complex DAML+OIL constructs can be used if required. The next several sections will show the examples we used in our matchmaking process.

5.2 Advertisement

Here we show an example of an advertisement. Suppose that we want to specify the concept of an advertisement by which the Actor would like to sell some PCs. In particular, there are some restrictions on the `Sales` and the `Delivery` such as the following:

- 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 the 15/09/2002;
- goods must be delivered in Bristol.

In DL notation, this advertisement can be written as:

$$
\begin{array}{lll}
Advert1 & \equiv & ServiceProfile \sqcap \\
 & & (\forall providedBy.(Actor \sqcap \forall hasName.\{Georgia\}) \sqcap \\
 & & \forall requestedBy.(Actor \sqcap \geq_5 \ hasCreditLevel) \sqcap \\
 & & \forall item.(PC \sqcap \geq_{128} \ memorySize) \sqcap \\
 & & \geq_{700} \ hasUnitPrice \sqcap \\
 & & \leq_{200} \ hasQuantity \sqcap \\
 & & \forall delivery.(Delivery \sqcap \\
 & & \leq_{20030501} \ date \sqcap \forall location.Manchester))
\end{array}
$$

In DAML+OIL, the way to express a concept like “has unit price more than 700” is to define a new datatype “more700” and describe it like “ $\forall\ hasUnitPrice.more700$”. However, for the purpose of achieving concept reasoning with datatypes using the RACER reasoner, we have used RACER syntax to express this kind of concept, e.g., “$(\geq_{700}\ hasUnitPrice)$”.

In addition, intuitively, an advertisement should be an instance instead of a concept, i.e., it looks more reasonable to express it as $Advert1 \in ServiceProfile \sqcap \cdots$. The reason of 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. What we did is to simply treat instance advertisements as atomic primitive concepts, hence, instead of having the ABox assertion $a : C$, TBox assertions like $C_a \sqsubseteq C$[6] is used. In contexts such as our e-commerce application, where individuals are not related to each other via properties, this does not lead to any loss of inferential power. This issue is, however, beyond the scope of this paper, and the interested reader is referred to [23].

5.3 Query

Similarly to the advertisement, we can define a query by which the Actor would like to buy some PCs. E.g., 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;

[6]The idea is to create a so-called “pseudo-concept” named $\mathsf{C_a}$ as a sub-concept of C, thus an individual instantiation is expressed using a concept implication.

- 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 allows us to do this while still being able to find relevant matches.

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

5.4 Revised Design

As discussed in Section 5.1, in accordance with DAML-S the providing and requesting Actors have been included as the values of *providedBy* and *requestedBy* properties in both the definition of advertisements and queries. It looks reasonable and rational, but there is a fatal problem lying in this design.

Consider the advertisement Advert1 in Section 5.2, which has the property $providedBy.(Actor \sqcap \forall hasName.\{Georgia\})$. Consider the request Query1 in Section 5.3, and suppose that we perform a matchmaking operation between Advert1 and Query1 using a DL reasoner to (semantically) compare the DAML+OIL descriptions. Due to the existence of $providedBy.(Actor \sqcap \forall hasName.\{Georgia\})$, there is no subsumption relationship between Query1 and Advert1: all we can do is prove that the two descriptions are not incompatible (their intersection is not equivalent to the bottom concept). This is always likely to be the case as the requester could not have the knowledge that the service he is looking for will be provided by an Actor with the name “Georgia”. This kind of match is quite weak, and does not allow for result selection via a hierarchy of match types with varying specificity (this will be discussed in detail in Section 6.2).

We believe that this design problem is inherent in 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.

In order to fix this problem, we have modified 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 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 like item information, unit price, unit quantity and delivery information. Later, in the matchmaking phase, we will only use this `ServiceProfile` part of an advertisement when computing semantic matches. Constraints such as *hasCreditLevel* might also be used in realistic e-commerce applications such as eBay[7] or Amazon,[8] but we do not consider them in our prototype.

In our modified design, we use the following notation to separate the different components of an advertisement:

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

and similarly we write queries as:

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

[7] `http://www.ebay.com/`

[8] `http://www.amazon.com/`

6 Matchmaking operation

6.1 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 advertisements[9] 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, $matches(Q)$:

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

Two descriptions are compatible if their intersection is satisfiable:

$$satisfiable(D1, D2) \Leftrightarrow \neg(D1 \sqcap D2 \sqsubseteq \bot)$$

For example, consider the following query:

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

The intersection of this query with `Advert1` in Section 5.4 is satisfiable. Formally:

$$Advert1 \in matches(Query2)$$

6.2 Matching Algorithm

To understand the matching algorithm we adopted in our prototype, we first need 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

[9]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, e.g., an advertiser might want to know the advertisements made by the others so that he can make some modification to his business strategy.

query are not semantically incompatible. Therefore, starting from the matching degree definition described in [20], we extend the match level "intersection satisfiable" to:

- **Exact** If advertisement A and request R are equivalent concepts, we call the match Exact; formally, $A \equiv R$.
- **PlugIn** If request R is sub-concept of advertisement A, we call the match PlugIn; formally, $R \sqsubseteq A$.
- **Subsume** If request R is super-concept of advertisement A, we call the match Subsume; formally, $A \sqsubseteq R$.
- **Intersection** If the intersection of advertisement A and request R is satisfiable, we call the match Intersection in order to distinguish it from Disjoint, where the advertisement and request are completely incompatible (this distinction was not made in [20]); formally, $\neg(A \sqcap R \sqsubseteq \bot)$.
- **Disjoint** Otherwise, we call the match Disjoint; that is, $A \sqcap R \sqsubseteq \bot$.

Degrees of the match are organized in a discrete scale. Exact matches are clearly preferable; PlugIn matches are considered the next best, since we might expect that advertisers also provide some more specific (sub-class) services, e.g., an advertiser selling PCs might be expected to sell some more specific kinds of PC; Subsume matches are considered to be third best, since an advertiser might also provide some more specific (super-class) services, e.g., an advertiser selling used PCs might also sell PCs in general;[10] Intersection is considered to be fourth best—it only says that the advertisement is not incompatible with the request; and Disjoint is the lowest level, since 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, we now introduce the process of matching a request. 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 request's ServiceProfile $\mathcal{R}$, i.e., to compute the input/output parts of $\mathcal{R}$'s subsumption relationships w.r.t. the input/output parts of all the advertisement ServiceProfiles. To express it more precisely, we present

[10]It could be argued that Subsume is preferable to PlugIn, but this discussion is beyond the scope of our work and would not qualitatively affect the performance of the prototype.

a piece of pseudocode in Table 1. In the pseudocode, the inequations like "degreeMatch $<$ globalDegreeMatch" follows the definition of match ordering, i.e., "Disjoint $<$ Intersection $<$ Subsume $<$ PlugIn $<$ Exact".

```
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
}
```

Table 1: Pseudocode for Request Matching

7 Prototype Implementation

In this section we describe the implementation of a multi-agent system including 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, as they were not considered relevant to our purpose: the investigation of ontology based service description and a DL based matchmaking service.

7.1 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 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 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.

7.2 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.

7.2.1 Advertising

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

7.2.2 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. There are two kinds of queries that can be defined:

- **Volatile Query:** the seeker submits a query to the Host, the matched advertisements are immediately returned, and then this query is discarded by the Host.

- **Persistent Query:** the seeker can also submit a persistent query to the Host. The persistent query is a query that will remain valid for a length of time defined by the Seeker itself. 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 have been added. The persistent query is automatically removed when the validity period is ended.

7.2.3 Modifying/Withdrawing

An Advertiser can modify and withdraw the advertisements it has published before. After the advertiser published his 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 we simply assume that all the partners in this framework are trusted.

7.2.4 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 that they will submit in turn, so as to maximize the likelihood of matching.

7.3 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 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 in Section 7.4 below.

- **AdvertiserAgent** publishes 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.

7.4 Matchmaking

At the beginning of the matchmaking process, the HostAgent initializes RACER with the service ontology described in Section 5, which the 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 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.

When it receives 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 in Section 6.2. 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 as classifying 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.

8 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, in order to find a match for a particular request, the RACER reasoner needs to check the satisfiability of the request with each advertisement that has been previously published to the Matchmaking host, and given the high worst case complexity of reasoning with DAML+OIL descriptions,[11] the question of scalability arises. We therefore used the prototype implementation 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 spent for the DL reasoner to find matched advertisements in response to a given request; the datasets are artificially generated by randomly creating the specifications of advertisements, e.g., the range of `location` and `hasProcessor` are randomly choosen from a set of concepts in the ontology, and the numbers of `memorySize` and `hasUnitPrice` are randomly choosen from a fixed range of integers; all the experiments have been 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.

Our 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 quite time-consuming. From the comparison of the different sized datasets shown in Figure 1, we can see that the average classification time per advertisement (shown on the Y axis) increases rapidly with the size of the dataset (shown on the X axis). The time rises from 49.57ms per advertisement for dataset size 100, and increases to 715.33ms per advertisement for dataset size 1,500.

Although this test illustrates that dataset size is an important issue in applications that use a DL reasoner, it does not mean that large datasets cannot be handled. For instance, in our prototype, we could do the TBox classification off-line, i.e., for all the published advertisements, we classify the TBox before the matchmaking process starts, and use the classified TBox to reason about requests. As to new incoming advertisements, we can simply insert them into the classified TBox hierarchy, which is *much* easier than classifying the entire TBox.[12]

[11]Key inference problems for the logic implemented in the RACER system have worst case ExpTime complexity in the size of the input.

[12]Although removing advertisements from the TBox hierarchy would be more difficult.

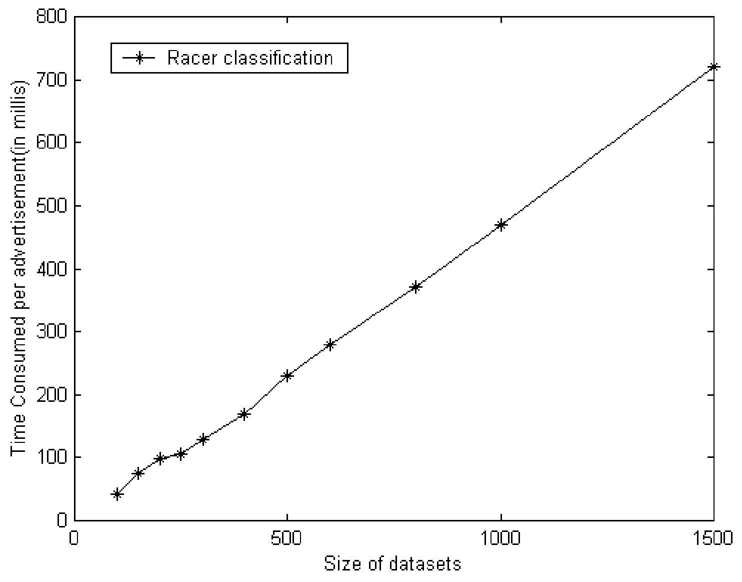

Figure 1: RACER classification times

9 Discussion

In this paper we have introduced service matchmaking in e-commerce, assessed the requirements for a service description language and ontology, and argued that DAML+OIL and DAML-S fulfill these requirements. This argument is supported by our design and implementation of a prototype matchmaker which 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 behaviour 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. We solved this problem 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 initial classification of large numbers of advertisements could be quite time consuming, subsequent matching of queries to advertisements could be performed very efficiently. On the basis of these preliminary results, it seems possible that DL reasoning technology could cope with large scale e-commerce applications; future work will include more extensive testing in order to clarify if this is, in fact, the case.

10 Acknowledgements

The author would like to thank 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 is funded by the Departmental Scholarship in the Department of Computer Science at the University of Manchester.

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 University Press, 2003.

[2] Bartolini, C.; Preist, C.; and Jennings, N. Architecting for reuse: A software framework for automated negotiation. In Giunchiglia, F., Odell, J., and Weiß, G. eds. *Proc. of the 3rd Int Workshop on Agent-Oriented Software Engineering*, pages 88–100. Springer, 2002.

[3] Bechhofer, S.; Horrocks, I.; Goble, C.; and Stevens, R. OilEd: A Reasonable ontology editor for the semantic web. In *Proc. of the Joint German/Austrian Conf. on Artificial Intelligence (KI 2001)*, number 2174 in Lecture Notes in Artificial Intelligence, pages 396–408. Springer, 2001. Appeared also in Proc. of the 2001 Description Logic Workshop (DL 2001).

[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 Recommentation. Available at `http://www.w3.org/TR/owl-ref/`, accessed on Dec. 2003.

[5] Bellifemine, F.; Poggi, A.; and Rimassa, G. JADE - a FIPA2000-compliant agent development environment. In *Proceedings of the fifth international conference on Autonomous agents*, pages 216–217. ACM, 2001.

[6] Berners-Lee, T. *Weaving the Web*. Harpur, San Francisco, 1999.

[7] Brickley, D. and Guha, R.V. RDF vocabulary description language 1.0: RDF schema. W3C Proposed Recommentation. Available at `http://www.w3.org/TR/rdf-schema/`, accessed on Dec. 2003.

[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):76–90, 1990.

[9] Haarslev, V. and Möller, R. RACER system description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*, volume 2083 of *Lecture Notes in Artificial Intelligence*, pages 701–705. Springer, 2001.

[10] Horrocks, I. DAML+OIL: a reason-able web ontology language. In Jensen, C.S., Jeffery, K.G., Pokorný, J., Saltenis, S., Bertino, E., Böhm, K. and Jarke, M. eds. *Proc. of EDBT 2002*, number 2287 in Lecture Notes in Computer Science, pages 2–13. Springer, Mar. 2002.

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

[12] Horrocks, I.; Patel-Schneider, P.F.; and van Harmelen, F. Reviewing the design of DAML+OIL: An ontology language for the semantic web. In *Proc. of the 18th Nat. Conf. on Artificial Intelligence (AAAI 2002)*, pages 792–797. AAAI Press, 2002.

[13] Horrocks, I. and Sattler, U. Ontology reasoning in the $\mathcal{SHOQ}$(D) description logic. In Nebel, B. ed. *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, pages 199–204. Morgan Kaufmann, 2001.

[14] Horrocks, I.; Sattler, U.; and Tobies, S. Practical reasoning for expressive description logics. In Ganzinger, H., McAllester, D. and Voronkov, A. eds. *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.

[15] Horrocks, I.; Sattler, U.; and Tobies, S. Reasoning with individuals for the description logic $\mathcal{SHIQ}$. In McAllester, D. ed. *Proc. of the 17th Int. Conf. on Automated Deduction (CADE 2000)*, volume 1831 of *Lecture Notes in Computer Science*, pages 482–496. Springer, 2000.

[16] McGuinness, D.L. Ontological issues for knowledge-enhanced search. In *Proceedings of Formal Ontology in Information Systems*, 1998. Also published in *Frontiers in Artificial Intelligence and Applications*, IOS-Press, 1998.

[17] Nodine, M.; Bohrer, W.; and Ngu, A. Semantic multibrokering over dynamic heterogeneous data sources in infosleuth. In *Proc. of the 15th International Conference on Data Engineering*, pages 358–365. IEEE Computer Society, 1999.

[18] 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):3–28, 2000.

[19] Noia, T.D.; Sciascio, E.D.; Donini, F.M.; and Mongiello, M. A system for principled matchmaking in an electronic marketplace. *International Journal of Electronic Commerce*, 2004. This volume.

[20] Paolucci, M.; Kawamura, T.; Payne, T.; and Sycara, K. Semantic matching of web services capabilities. In Horrocks, I. and Hendler, J. eds. *Proc. of the 1st International Semantic Web Conference (ISWC)*, pages 333–347. Springer, 2002.

[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):47–53, 1999.

[22] Sycara, K.; Paolucci, M.; van Velsen, M.; and Giampapa, J. The RETSINA MAS infrastructure. Technical Report CMU-RI-TR-01-05, Robotics Institute Technical Report, Carnegie Mellon, 2001.

[23] Tessaris, S. *Questions and Answers: Reasoning and Querying in Description Logic*. PhD thesis, University of Manchester, Department of Computer Science, Apr. 2001.

[24] The DAML Services Coalition. Daml-s: Semantic markup for web services. Available at http://www.daml.org/services/daml-s/0.9/daml-s.html, accessed on May 2003.

[25] Trastour, D.; Bartolini, C.; and Preist, C. Semantic web support for the business-to-business e-commerce lifecycle. In *Proceedings of the Eleventh International conference on World Wide Web*, pages 89–98. ACM, 2002.

[26] UDDI technical white paper. Available at http://www.uddi.org/, accessed on Nov. 2002.

[27] van Harmelen, F.; Patel-Schneider, P.F.; and Horrocks, I. Reference description of the DAML+OIL (March 2001) ontology markup langauge. Available at http://www.daml.org/2001/03/reference.html, accessed on Nov. 2002.

[28] Web services description language (WSDL) 1.1. W3C note 15 march 2001. Available at http://www.w3.org/TR/wsdl/, accessed on Nov. 2002.