Matchmaking among Heterogeneous Agents on the Internet*

Katia Sycara, Jianguo Lu†, Matthias Klusch, Seth Widoff
The Robotics Institute, Carnegie Mellon University, Pittsburgh, USA.
{katia, klusch, swidoff}@cs.cmu.edu

Abstract

The Internet not only provides data for users to browse, but also databases to query, and software agents to run. Due to the exponential increase of deployed agents on the Internet, automating the search and selection of relevant agents is essential for both users and collaboration among different software agents. This paper first describes the agent capability description language LARKS. Then we will discuss the matchmaking process using LARKS and give a complete working scenario. The paper concludes with comparing our language and the matchmaking process with related works. We have implemented LARKS and the associated powerful matchmaking process, and are currently incorporating it within our RETSINA multi-agent framework.

Keywords: Matchmaking, Agent Collaboration and Coordination

1 Introduction

The Internet not only provides data for users to browse in the Web, but also heterogeneous databases to query, and software agents to run. Due to the exponential increase of deployed agents on the Internet, automating the searching and selection of relevant agents is essential for both users and the software agent society in several ways. Firstly, novice users in Cyberspace may have no idea where to find a service, and what agents are available for performing a task. Secondly, even experienced users can't be aware of every change on the Internet. Relevant agents may appear and disappear over time. Thirdly, as the number and sophistication of agents on the Internet increase, there is an obvious need for standardized, meaningful communication among agents to enable them to perform collaborative task execution.

To facilitate the searching and interoperation between agents on the Internet, we proposed the RETSINA multi-agent infrastructure framework[18]. In this framework, we distinguished two general agent categories: service providers and service requester agents. Service providers provide some type of service, such as finding information, or performing some particular domain specific problem solving (*e.g.*, number sorting). Requester agents need provider agents to perform some service for them. Since the Internet is an open environment where information sources, communication links, and agents themselves may appear and disappear unpredictably, there is a need for some means to help requester agents find providers. Agents that help locate other agents are called *middle agents*.

We have identified different types of middle agents on the Internet, such as matchmakers (yellow page services), brokers, billboards, etc., and experimentally evaluated different protocols for interoperation between providers, requesters, and various types of middle agents[2]. We have also developed protocols for distributed matchmaking [7]. *Matchmaking* is the process of finding an appropriate provider through a middle agent, and has the following general form (Figure 1):

- Provider agents advertise their capabilities to middle agents.
- Middle agents store these advertisements.

*This research has been sponsored in part by Office of Naval Research grant N-00014-96-16-1-1222.
†Department of Computer Science, University of Toronto, Canada. Email: jglu@cs.toronto.edu

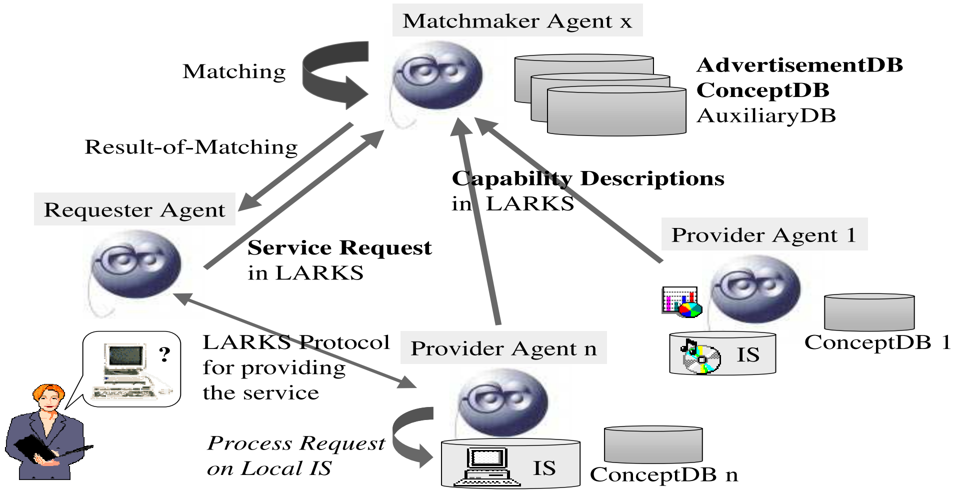

Figure 1: Matchmaking using LARKS: An Overview

- A requester asks some middle agent whether it knows of providers with desired capabilities.
- The middle agent matches the request against the stored advertisements and returns the result, a subset of the stored advertisements.

While this process at first glance seems very simple, it is complicated by the fact that providers and requesters are usually heterogeneous and incapable of understanding each other. This difficulty gives rise to the need for a common language for describing the capabilities and requests of software agents in a convenient way. In addition, one has to devise a mechanism for matching descriptions in that language. This mechanism can then be used by middle agents to efficiently select relevant agents for some given tasks.

In the following, we first describe the agent capability description language, LARKS. Then we will discuss the matchmaking process using LARKS and give a complete working scenario. The paper concludes with comparing our language and the matchmaking process with related works. We have implemented LARKS and the associated powerful matchmaking process, and are currently incorporating it within our RETSINA multi-agent infrastructure framework [18].

2 The Agent Capability Description Language LARKS

2.1 Desiderata for an Agent Capability Description Language

There is an obvious need to describe agent capabilities in a common language before any advertisement, request, or even matchmaking among the agents can take place. In fact, the formal description of capabilities is one of the difficult problems in the area of software engineering and AI. Some of the main desired features of such a agent capability description language (ACDL) are the following:

- **Expressiveness** The language should be expressive enough to represent not only data and knowledge, but also the meaning of program code. Agent capabilities should be described at an abstract rather than implementation level. Most existing agents should be distinguishable by their descriptions in this language.
- **Inferences**. Inferences on descriptions written in this language should be supported. Automated reasoning and comparison on the descriptions should be possible and efficient.

- **Ease of Use**. Descriptions should not only be easy to read and understand, but also easy to write by the user. The language should support the use of domain or common ontologies for specifying agents capabilities.

- **Application in the Web**. One of the main application domains for the language is the specification of advertisements and requests of agents on the Web. The language allows for automated exchange and processing of information among these agents.

There are many program description languages, like Z[13], to describe the functionalities of programs. These languages contain too much detail to be useful for the searching purpose. Also reading and writing specifications in these languages require sophisticated training. On the other hand, the interface definition languages, like IDL and WIDL, go to the other extreme by completely omitting the functional descriptions of the services. Only the input and output information is provided.

In AI, knowledge description languages like KIF are meant to describe the knowledge instead of the actions of a service. The action representation formalisms like STRIPS are too restrictive to represent complicated services. Some agent communication languages like KQML and FIPA ACL concentrate on the communication protocals (message types) between agents but leave the content part of the language unspecified.

In Internet computing, various description formats are being proposed, notably the WIDL and the Resource Description Framework(RDF)[14]. Although the RDF also aims at the interoperablity between Web applications, it is intended to be a basis for describing metadata. RDF allowes different vendors to describe the properties and relations between resources on the Web. That enables other programs, like Web robots, to easily extract relevant information, and to build a graph structure of the resources available on the Web, without the need to give any specific information. However, the description does not describe the functionalities of the Web *services*.

Since none of those languages satisfies our requirements, we propose an ACDL, called LARKS (**L**anguage for **A**dvertisement and **R**equest for **K**nowledge **S**haring), that enables for advertising, requesting and matching agent capabilities.

2.2 Specification in LARKS

A specification in LARKS is a frame with the following slot structure.

`Context`	Context of specification
`Types`	Declaration of used variable types
`Input`	Declaration of input variables
`Output`	Declaration of output variables
`InConstraints`	Constraints on input variables
`OutConstraints`	Constraints on output variables
`ConcDescriptions`	Ontological descriptions of used words

The frame slot types have the following meaning.

- `Context` The context of the specification in the local domain of the agent.

- `Types` Optional definition of the data types used in the specification.

- `Input` and `Output` Input/output variable declarations for the specification. In addition to the usual type declarations, there may also be concept attachments to disambiguate types of the same name. The concept itself is defined in the concept description slot `ConcDescriptions`.

- `InConstraints` and `OutConstraints` Logical constraints on input/output variables that appear in the input/output declaration part. The constraints are described as Horn clauses.

- `ConcDesriptions` Optional description of the meaning of words used in the specification. The description relies on concepts in a given local domain ontology. Attachement of a concept C to a word w in any of the slots above is done in the form: w*C. That means that the concept C is the ontological description of the word w. The concept C is included in the slot `ConcDescription`.

Every specification in LARKS can be interpreted as an advertisement as well as a request; the specification's role depends on the agent's purpose for sending it to a matchmaker agent. Every LARKS specification must be wrapped by the sending agent in an appropriate KQML message[1] that indicates if the message content is to be treated as a request or an advertisement.

2.3 Using Domain Knowledge in LARKS

LARKS offers the option to use application domain knowledge in any advertisement or request. This is done by using a local ontology for describing the meaning of a word in a LARKS specification. Local ontologies can be formally defined using concept languages such as ITL.

The main benefits of providing a local ontology are twofold: (1) the user can specify in more detail what's being requested or advertised, and (2) the matchmaker agent is able to make automated inferences on these additional semantic descriptions while matching LARKS specifications, thereby improving the overall quality of the matching process.

Example 2.1: *Finding informations on computers*

Suppose that a provider agent, such as HotBot or Excite, advertises the capability to find information about any type of computer. The administrator of the agent might specify that capability in LARKS as follows:

FindComputerInfo	
`Context`	Computer*Computer;
`Types`	Model, Brand, Color, State, CPU = `String`; Price = `Real`; InfoList = `ListOf`(model: Model*ComputerModel, brand: Brand*Brand, price: Price*Money, color: Color*Colors);
`Input`	brands: `SetOf` Brand*Brand; areas: `SetOf` State; processor: `SetOf` CPU*CPU; priceLow*LowPrice: `Integer`; priceHigh*HighPrice: `Integer`;
`Output`	Info: InfoList;
`InConstraints`	
`OutConstraints`	sorted(Info);
`ConcDescriptions`	Computer = (and Product (`exists` has-processor CPU) (`all` has-memory Memory) (`all` is-model ComputerModel)); LowPrice = (and Price (`ge` 1800)(`exists` in-currency `aset`(USD))); HighPrice = (and Price (`le` 50000)(`exists` in-currency `aset`(USD))); ComputerModel = `aset`(HP-Vectra,PowerPC-G3,Thinkpad770,Satellite315); CPU = `aset`(Pentium,K6,PentiumII,G3,Merced) [Product, Colors, Brand, Money]

○

Most words in this specification have been attached with a name of some concept out of the supplied ontology, whose concept definitions are included in the slot `ConcDescriptions`. Concept definitions

[1] Although the current implementation supports agent messages in KQML, LARKS is independent of any communicztion language or set of performatives/speech acts.

which were already sent to the matchmaker are enclosed in brackets. In this example we assume the underlying ontology to be written in the concept language ITL[17].

3 The Matchmaking Process Using LARKS

3.1 Different Types of Matching in LARKS

Agent capability matching is the process of determining whether an advertisement registered in the matchmaker matches a request. Before we go into the details of the matchmaking process, we should clarify the ways in which two specifications can match.

- **Exact Match** The most accurate type of matching is when both descriptions are equivalent, by being literally equal, equal when the variables are renamed, or equal by logical inference. This type of matching is the most restrictive one.

- **Plug-In Match** A less accurate but most useful type of match is the *plug-in* match. In a plug-in match, the advertised input types are not more constrained than the requested input types, and the advertised output types are not less constrained than the requested output types. A service described by a plug-in advertisement for a request will not demand more or more specific input parameters than the requester can supply, and will not return fewer or more general output parameters than the requester needs. Hence the requester submits a subset of its input paratemers to get in return a superset of the output parameters it needs. Exact match is a special case of plug-in match, that is, wherever two descriptions are an exact match, they are also a plug-in match.

  A simple example of a plug-in match is between a request to sort a list of integers and an advertisement of an agent that can sort both lists of integers or strings. This example is elaborated on in section 4.

- **Relaxed Match** The least accurate type of match is the *relaxed* match. A relaxed matching process will not return advertised services that can be immediately used by the requester, but instead it returns the subset of advertisements whose distance from the request is less than some supplied threshold. Thus the requester obtains a survey of the kinds of similar advertised services that have been registered with the matchmaker.

  For a request describing a service to locate Compaq computer dealers in a state, a relaxed match might return an advertisement describing a service that returns a dealer location and price given a Compaq computer model.

Different users in different situations may want to employ different types of matches. Thus the matchmaker agent supports several kinds of filters, which have different criteria for determining if advertisements and requests match.

3.2 The Filtering Stages of the Matchmaking Process

The matching engine of the matchmaker agent contains five different filters:

1. *Context matching*,
2. *Profile comparison*,
3. *Similarity matching*,
4. *Signature matching*, and
5. *Constraint matching*.

The first three filters are meant for relaxed matching, and the signature and constraint matching filter are meant for plug-in matching [2]. On the basis of the given notions of matching we implemented four different modes of matching for the matchmaker:

1. **Complete Matching Mode**. All filters are considered.
2. **Relaxed Matching Mode**. The context, profile, and similarity matching is done.
3. **Profile Matching Mode**. Only the context matching and comparison of profiles is performed.
4. **Plug-In Matching Mode**. In this mode, the matchmaker performs the signature and constraint matching only.

Users may select any of these modes or combination of these filters on demand. For example, when efficiency is the major concern, a user might select only the context and profile filters. On the other hand, when a user or agent wants to find agents that plug-in match, then the matchmaking process would be configured with signature and constraint filters. We will now describe each filter in more detail.

3.2.1 Context Filter

Any matching of two specifications has to be in an appropriate context. In LARKS to deal with restricting the advertisement matching space to those in the same domain as the request, each specification supplies a list of key words meant to describe the domain of the service.

When comparing two specifications it is assumed that their domains are the same (or at least sufficiently similar) as long as (1) the real-valued distances between the roots of these words do not exceed a given threshold and (2) subsumption relations between the attached concepts of the most similar words are the same. The matching process only proceeds if both conditions hold.

3.2.2 Profile Filter

Although context matching is efficient, it does not consider the whole specification itself. This is done with a profile filter that compares two LARKS specifications by using the TF-IDF (term frequency-inverse document frequency)[16] technique from the domain of information retrieval.

Each specification in LARKS is treated as a document, and each word w in a document Req is weighted for that document in the following way. The number of times w occurs throughout all documents in the matchmaker's advertisement database is called the document frequency — $df(w)$ — of w.

Thus for a given document d, the relevance of d for word w is proportional to the number of times w occurs in d — $wf(w,d)$ — and inversely proportional to $df(w)$. A weight $h(w,d)$ for a word in a document d out of a set D of documents denotes the significance of the classification of w for d, and is defined as follows:

$$h(w,d) = wf(w,d) \cdot log(\frac{|D|}{df(w)}).$$

The weighted keyword representation $wkv(d,V)$ of a document d contains the weight $h(w,d)$ as an element for every word w in a given dictionary V. Since most dictionaries provide a huge vocabulary, we cut down the dimension of the vector by using a fixed set of appropriate keywords determined by heuristics and the set of keywords in LARKS itself.

The similarity $dps(Req, Ad)$ of a request Req and an advertisement Ad under consideration is then calculated by :

$$dps(Req, Ad) = \frac{Req \bullet Ad}{|Req| \cdot |Ad|}$$

where $Req \bullet Ad$ denotes the inner product of the weighted keyword vectors. If the value $dps(Req, Ad)$ exceeds a given threshold $\beta \in \mathbf{R}$, the matching process continues with the following steps.

[2] The computational costs of these filters are in increasing order.

3.2.3 Similarity Filter

The profile filter has two drawbacks. First, it does not consider the structure of the description. That means the filter, for example, is not able to differentiate among input and output declarations of a specification. Second, profile comparison does not rely on the semantics of words in a document. Thus the filter is not able to recognize that the word pair (Computer, Notebook), for example, should have a closer distance than the pair (Computer, Book).

Computation of similarity distance is a combination of distance values as calculated for pairs of input and output declarations, and input and output constraints. Each of these distance values is computed in terms of the distance between concepts and words that occur in their respective specification section. The values are computed at the time of advertisement submittal and stored in the matchmaker database. Word distance is computed using the trigger-pair model [15]. If two words are significantly co-related, then they are considered trigger-pairs, and the value of the co-relation is domain specific. In the current implementation we use the Wall Street Journal corpus of one million word pairs to compute the word distance. The distance computation between concepts is discussed in section 3.3.

3.2.4 Signature and Constraint Filters

The similarity filter takes into consideration the semantics of individual words in the description. However, it does not take the meaning of constraints in a LARKS specification into account. A more sophisticated semantical matching is needed. This is done in our matchmaking process by the signature and constraint filters. The two filters are designed to work together to look for a so-called plug-in match.

Signature matching checks if the signatures of input and output declarations match. It is performed by a set of subtype inference rules as well as concept subsumption testing (see [17] for details).

In software engineering it is proven that a component description Desc2 'plug-in matches' another description Desc1 if

- Their signatures match.
- InConstraint of Desc1 implies the InConstraints of Desc2, i.e., for every clause $C1$ in the set of input constraints of $Spec1$ there is a clause $C2$ in the set of input constraint of $Spec2$ such that $C1 \preceq_\theta C2$.
- OutConstraints of Desct2 implies the OutConstraints of Desc1, i.e., for every clause $C2$ in the set of output constraints of $Spec2$ there is a clause $C1$ in the set of output constraints of $Spec1$ such that $C2 \preceq_\theta C1$.

where $\preceq_\theta$ denotes the θ-subsumption[12] relation between definite program clauses.

The main problem in performing plug-in matching is that the logical implication between constraints is not decidable for first order predicate logic, and not even for an arbitrary set of Horn clauses. To make the matching process tractable and feasible, we decided to use θ-subsumption, a relation that is weaker than logical implication.

3.3 Concept Subsumption Checking

Concept subsumption relationships and concept distances are frequently used in the matching engine, especially in the similarity, signature, and constraint filters. These relations are computed at the time each advertisement is submitted and stored in the Concept Database.

A concept C subsumes another concept C' if the extension of C' is a subset of C. This means that the logical constraints defined in the term of the concept C' logically imply those of the more general concept C.

Any concept language is decidable if concept subsumption between any two concepts defined in that language can be determined. The concept language ITL we use is NP-complete decidable. We use an incomplete inference algorithm for computing subsumption relations between concepts in ITL [3]. For the mechanism of subsumption computation refer to [19].

3.4 Computation of Distances Among Concepts

For matchmaking, the identification of relations other than subsumption between concepts is very useful because it promotes a deeper semantic understanding. Moreover, since we've restricted the expressiveness of the concept language ITL in order to boost performance, we need some way to express additional associations between concepts.

To express these associations we use a weighted associative network (AN), a semantic network with directed edges between concept nodes. The type of edge between two concepts denotes their binary relation, and edges are labeled with a numerical weight (interpreted as a fuzzy number). The weight indicates the strength of belief in that relation, since its real world semantics may vary.

In our implementation we created an associative network by using the concept subsumption hierarchy and additional associations set by the user. Distance between two concepts C, C' in an AN is computed as the strength ofthe shortest path between C and C' on the basis of triangular norms (see [3] for details).

4 Example of Matchmaking Using LARKS

Consider the following simple specifications 'IntegerSort' and 'GenericSort', a request for an agent that sorts integer numbers, and an advertisement for an agent that is capable of sorting real numbers and strings.

IntegerSort	
`Context`	Sort
`Types`	
`Input`	xs: `ListOf Integer`;
`Output`	ys: `ListOf Integer`;
`InConstraints`	le(length(xs),100);
`OutConstraints`	before(x,y,ys) $<-$ ge(x,y); in(x,ys) $<-$ in(x,xs);
`ConcDescriptions`	

GenericSort	
`Context`	Sorting
`Types`	
`Input`	xs: `ListOf` Real | String;
`Output`	ys: `ListOf` Real | String;
`InConstraints`	

`OutConstraints`	before(x,y,ys) $<-$ ge(x,y);
	before(x,y,ys) $<-$ preceeds(x,y);
	in(x,ys) $<-$ in(x,xs);

`ConcDescriptions`	

Assume that the provider agent submits the advertisement 'GenericSort' to the matchmaker, and that later the requester submits the request 'IntegerSort'.

Context Matching
Both words in the `Context` declaration parts are sufficiently similar since they share the same root. We have no referenced concepts to check for terminologically equity, thus the matching process proceeds with the following two filtering stages.

Comparison of Profiles
According to the result of the TF-IDF procedure both specifications are sufficiently similar.

Similarity Matching
Using the current auxiliary database for word distance values, similarity matching of constraints yields:

le(length(xs),100))	null	= 1.0
before(x,y,ys) $<-$ ge(x,y)	in(x,ys) $<-$ in(x,xs)	= 0.5729
in(x,ys) $<-$ in(x,xs)	before(x,y,ys) $<-$ preceeds(x,y)	= 0.4375
before(x,y,ys)$<-$ ge(x,y)	before(x,y,ys) $<-$ preceeds(x,y)	= 0.28125

[3] Using the well-known tradeoff, we compromise expressiveness for tractability in our subsumption algorithm, which is correct but incomplete.

The similarity of both specifications is computed as:
$Sim(\texttt{IntegerSort}, \texttt{GenericSort}) = 0.64$.

Signature Matching
Consider the signatures t_1= (ListOf Integer) and t_2= (ListOf Real|String). Following the subtype inference rules 9., 4., and 1., it holds that $t_1 \preceq_{st} t_2$, but not vice versa, thus the signature of D_{11} is subsumed by that of D_{21}). The same with D_{12} and D_{22}.

Constraint Matching
The advertisement 'GenericSort' is also a plug-in match semantically with the request 'IntegerSort', because the input constraints of 'IntegerSort' are θ-subsumed by those of 'GenericSort', and the output constraints of 'GenericSort' are θ-subsumed by those of 'IntegerSort'. Thus 'GenericSort' plugs into 'IntegerSort'. Please note that the reverse does not hold.

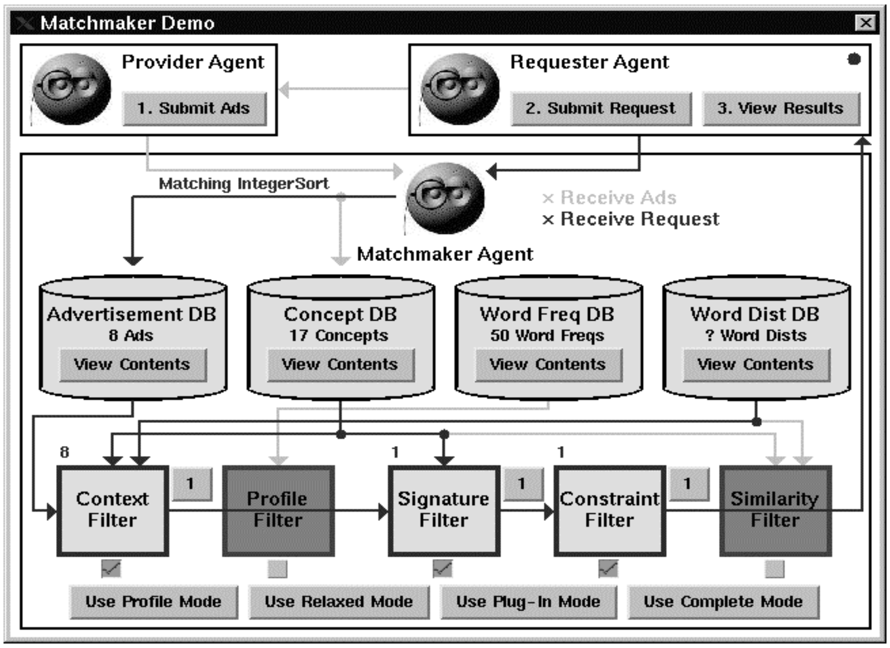

Figure 2: The User Interface of the Matchmaker Agent.

Figure 2 shows the user interface of the implemented matchmaker agent. To help visualize the matchmaking process, we devised a user interface that traces the path of the advertisement result set for a request through the matchmaker's filters. The filters can be configured by selecting the checkboxes beneath the desired filters — disabled filters are darkened and bypassed — or by pressing one of the buttons for a predefined mode. As the result set passes from one filter to the next, the filter's outline highlights, the number above the filter increments as it considers an advertisement, and the number above its output arrow increments as advertisements successfully pass through the filter. Pushing the buttons above each inter-filter arrow reveals the result advertisement set for the preceding filter.

5 Related Work

Agent matchmaking has been actively studied since the inception of software agent research. The earliest matchmaker we are aware of is the ABSI facilitator, which is based on the KQML specification and uses the KIF as the content language. The KIF expression is basically treated like Horn clauses. The matching between the advertisement and request expressed in KIF is the simple unification with the equality predicate.

The SHADE and COINS[10] are matchmakers based on KQML. The content language of COINS allows for free text and its matching algorithm utilizes the TF-IDF like the profile filter. The context language of the SHADE matchmaker consists of two parts, one is a subset of KIF, the other is a structured logic representation called MAX. MAX use logic frames to declaratively store knowledge. SHADE uses a frame-like representation and matching relies on a Prolog-like unification process.

A more recent service broker-based information system is InfoSleuth[6]. The content language supported by InfoSleuth is KIF and the deductive database language is LDL++, which has semantics similar to Prolog. The constraints for both the user request and the resource data are specified in terms of some given central ontology. It is the use of this common vocabulary that enables the dynamic matching of requests to the available resources. The advertisements specify agents' capabilities in terms of one or more ontologies. The constraint matching is an intersection function between the user query and the data resource constraints. If the conjunction of all the user constraints with all the resource constraints is satisfiable, then the resource contains data that are relevant to the user request.

Broker and matchmaker agents can be seen as a kind of so-called mediator agents among heterogeneous information systems[21, 1]. Each local information system is 'wrapped' by a wrapper agent, and their capabilities are described in two levels. The first is what they can provide, which is usually described in the local data model and local database schema. The second is what kind of queries they can answer, which is usually a subset of SQL. The set of queries a service can accept is described using a grammar-like notation. Matching between the query and the service is simple: it just decides whether the query can be generated by this grammar. This area emphasizes the planning of database queries according to heterogeneous information systems not providing complete SQL services. Those systems are not designed to be searched for among a vast number of Internet resources. The description of capabilities and matching are not only studied in the agent community, but also in other related areas.

5.1 Works Related with Capability Description

The following approaches provide solutions for the problem of capability and service description matching:

1. Software specification techniques.
   Agents are computer programs that have some specific characteristics. There is numerous work on software specifications in formal methods, like the model-oriented VDM and Z[13], or the algebra-oriented Larch. Although these languages are good at describing computer programs in a precise way, the specification usually contains too many details to be of interest to other agents. The complexity of these languages prohibits effective semantic comparison between the specifications. Reading and writing these specifications also requires substantial training.

2. Action representation formalisms.
   Agent capability can be seen as the actions that the agents perform. There are a number of action representation formalisms in AI planning, like the classical one, STRIPS. Action representation formalisms are inadequate for our task since they are propositional and do not involve data types.

3. Concept languages for knowledge representation.
   There are various terminological knowledge representation languages. However, an ontology itself does not describe any capability. On the other hand, it provides auxiliary concepts to assist the specification of agent capabilities.

4. Database query capability description.

   The database query capability description technique in [21] was developed as an attempt to describe the information sources on the Internet, such that an automated integration of information would be possible. In this approach the information source is modeled as a database with restricted querying capabilities.

5.2 Works Related to Service Retrieval

In software component search techniques, [22] defines several notions of matching, including exact and plug-in matching, and formally proves the relationship between those matches. [5] proposes to use a sequence of filters to search for software components to increase the efficiency of the search process. [8] computes the distance between similar specifications. This work is based on algebraic specification of computer programs. No concept descriptions and concept hierarchies are considered.

In Web resource search techniques, [11] proposes a method to look for better search engines that may provide more relevant data for the user concerns, and ranks those search engines according to their relevance to a user's query. They propose a directory of services to record descriptions for each information server. A user sends a query to a directory of services, which determines and ranks the servers that are relevant to the user's request. Both the query and the server are described using boolean expressions. The search method is based on the similarity measure between the two boolean expressions.

6 Conclusion

The Internet is an open system where heterogeneous agents can appear and disappear dynamically. As the number of agents on the Internet increases, there is a need to define middle agents to help agents locate other agents that provide requested services. In prior research we have identified a variety of middle agent types, their protocols and their performance characteristics. Matchmaking is the process that brings requester and service provider agents together. A provider agent advertises its know-how or capabilities to a middle agent, which stores the advertisements. An agent that desires a particular service sends a service request to a middle agent, which subsequently matches the request against its stored advertisements. The middle agent communicates the results to the requester (the way this happens depends on the type of middle agent involved).

We have also defined protocols that allow more than one middle agent to maintain consistency in their advertisement databases. Since matchmaking is usually done dynamically and over large networks, it must be efficient. There is an obvious trade-off between the quality and efficiency of service matchmaking on the Internet.

We have defined and implemented a language called LARKS for agent advertisements and requests, and a matchmaking process that uses LARKS. LARKS judiciously balances language expressiveness and efficiency in matching. The LARKS matchmaking process performs both syntactic and semantic matching, and in addition allows the specification of concepts (local ontologies) via ITL, a concept language.

The matching process uses five filters: context matching, profile comparison, similarity matching, signature matching and constraint matching. Different degrees of partial matching can result from utilizing different combinations of these filters. The selection of filters to apply is under the control of the user (or the requester agent).

Acknowledgement: We thank Davide Brugali, Somesh Jha and Anandeep Pannu for their helpful discussions in this project.

References

[1] J.L. Ambite and C.A. Knoblock. Planning by Rewriting: Efficiently Generating High-Quality Plans. Proceedings of the Fourteenth National Conference on Artificial Intelligence, Providence, RI, 1997.

[2] K. Decker, K. Sycara, M. Williamson. Middle-Agents for the Internet. Proc. 15th IJCAI, pages 578-583, Nagoya, Japan, August 1997.

[3] P. Fankhauser, E.J. Neuhold. Knowledge based integration of heterogeneous databases. Proceedings of IFIP Conference DS-5 Semantics of Interoperable Database Systems, Lorne, Victoria, Australia, 1992.

[4] T. Finin, R. Fritzson, D. McKay, R. McEntire. KQML as an Agent Communication Language. Proc. 3rd International Conference on Information and Knowledge Management CIKM-94, ACM Press, 1994.

[5] J. Goguen, D. Nguyen, J. Meseguer, Luqi, D. Zhang, V. Berzins. Software component search. Journal of Systems Integration, 6, pp. 93-134, 1996.

[6] N. Jacobs, and R. Shea. The role of Java in InfoSleuth: Agent-based exploitation of heterogeneous information ressources. Proc. of Intranet-96 Java Developers Conference, April 1996.

[7] S. Jha, P. Chalasani, O. Shehory and K. Sycara. A Formal Treatment of Distributed Matchmaking. In Proceedings of the Second International conference on Autonomous Agents (Agents 98), Minneapolis, MN, May 1998.

[8] J-J. Jeng, and B.H.C. Cheng. Specification matching for software reuse: a foundation. Proceedings of the ACM SIGSOFT Symposium on Software Reusability, ACM Software Engineering Note, Aug. 1995.

[9] M. Kracker. A fuzzy concept network. Proc. IEEE International Conf. on Fuzzy Systems, 1992.

[10] D. Kuokka, L. Harrada, On using KQML for Matchmaking. Proc. 3rd Intl. Conf. on Information and Knowledge Management CIKM-95, pp. 239-45, AAAI/MIT Press, 1995.

[11] S.-H. Li, P. B. Danzig. Boolean Similarity Measures for Resource Discovery. IEEE Transactions on Knowledge and Data Engineering, Vol.9, No. 6, November/December, 1997.

[12] S. Muggleton, and L. De Raedt. Inductive logic programming: theory and methods. Journal of Logic Programming, 19,20:629–679, 1994.

[13] B. Potter, J. Sinclair, and D. Till. Introduction to Formal Specification and Z. Prentice-Hall International Series in Computer Science.

[14] Resource Description Framework (RDF) Schema Specification. http://www.w3.org/TR/WD-rdf-schema/.

[15] R. Rosenfield. Adaptive statistic language model. PhD thesis, Carnegie Mellon University, 1994.

[16] G. Salton, A. Wong. A vector space model for automatic indexing. Communications of the ACM, 18, 613-620, 1975.

[17] K. Sycara, J. Lu, and M. Klusch. Interoperability among Heterogeneous Software Agents on the Internet. Carnegie Mellon University, PA (USA), Technical Report CMU-RI-TR-98-22, 1998.

[18] K. Sycara, K. Decker, A. Pannu, M. Williamson, and D. Zeng. Distributed Intelligent Agents. IEEE Expert, pp36-46, December 1996.

[19] G. Smolka, and M. Schmidt-Schauss. Attributive concept description with complements. AI, 48, 1991.

[20] G. Wickler: Using Expressive and Flexible Action Representations to Reason about Capabilities for Intelligent Agent Cooperation. http://www.dai.ed.ac.uk/students/gw/phd/story.html

[21] V. Vassalos, Y. Yapakonstantinou. Expressive Capabilities Description Languages and Query Rewriting Algorithms. available at http://www-cse.ucsd.edu/ yannis/papers/vpcap2.ps

[22] A. M. Zaremski, J. M. Wing Specification matching of software components. Carnegie Mellon University, PA (USA), Technical Report CMU-CS-95-127, 1995.