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 is not only providing 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 infrastructure framework.

Keywords: Agent Interoperability, Matchmaking, Collaboration.

1 Introduction

Nowadays the Internet is not only providing 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 searching and selection of relevant agents is essential for both users and the software agent society in several ways. Firstly, novice users in the cyberspace may have no idea where to find the service, what agents are available for doing their job. Secondly, experienced users may not 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 a standardized, meaningful communication among agents to enable them to perform collaborative task execution.

To facilitate the searching and interoperation among agent on the Internet, we proposed the RETSINA multi-agent infrastructure framework[18]. In this framework, we distinguish 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 others 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]. The process of finding an appropriate provider through a middle agent is called *matchmaking*. It has the following general form (Figure 1):

- Provider agents advertise their capabilities such as know-how, expertise, and so on, 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

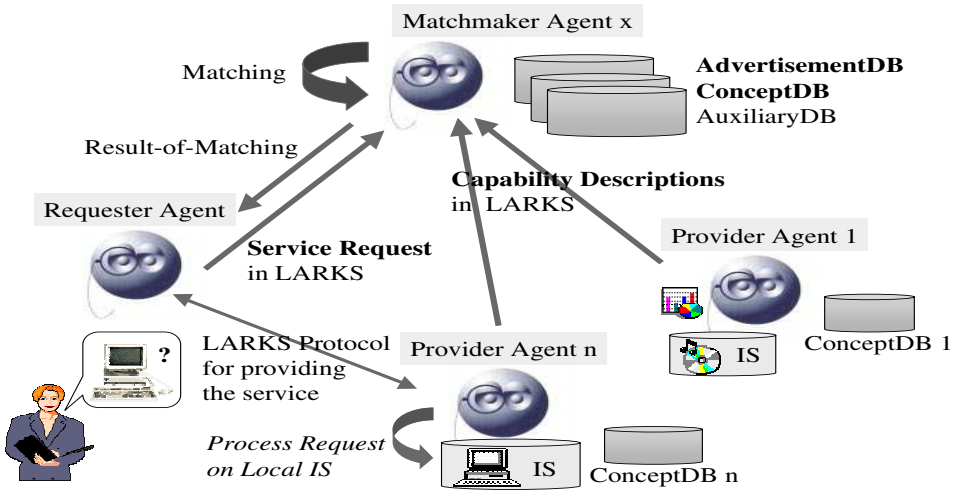

Figure 1: Matchmaking using LARKS: An Overview

- 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.

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 or users 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 of existing agents should be able to be distinguished 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 in 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 concern 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 omitting the functional descriptions of the services at all. Only the input and output information are 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 service. 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 format 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 rather 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 input and 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 declaration, there may have concept attachment. 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 restricted to be 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; this depends on the purpose for which an agent sends a specification to some matchmaker agent(s). Every LARKS specification must be wrapped up in an appropriate KQML message by the sending agent indicating 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 benefit of that option is twofold: (1) the user can specify in more detail what he is requesting or advertising, and (2) the matchmaker agent is able to make automated inferences on such kind of additional semantic descriptions while matching LARKS specifications, thereby improving the overall quality of matching.

Example 2.1: *Finding informations on computers*

Suppose that a provider agent such as, e.g., HotBot, Excite, advertises the capability to find informations about any type of computers. The administrator of the agent may specify that capability in LARKS as follows.

FindComputerInfo	
`Context`	Computer*Computer;
`Types`	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 a given ontology. The definitions of these concepts are included in the slot `ConcDescriptions`. Concept definitions 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 various notions of matches of two specifications.

- **Exact Match** The most accurate match is when both descriptions are equivalent, either equal literally, or equal by renaming the variables, or equal logically obtained by logical inference. This type of matching is the most restrictive one.
- **Plug-In Match** A less accurate but more useful match is the so-called $plug-in$ match. Roughly speaking, plug-in matching means that the agent which capability description matches a given request can be "plugged into the place" where that request was raised. As we can see, exact match is a special case of plug-in match, i.e., wherever two descriptions are exact match, they are also plug-in match.

  A simple example of a plug-in match is that of the match between a request to sort a list of integers and an advertisement of an agent that can sort both list of integers and list of strings. This example is elaborated in section 4.
- **Relaxed Match** The least accurate but most useful match is the so-called *relaxed* match. Relaxed match will not tell whether one description can be reused by another. Instead it determines how close the two descriptions are by returning a numerical distance value. Two descriptions match if the distance value is smaller than a preset threshold value.

  An example of a relaxed match is that of the request to find the place (or address) where to buy a Compaq Pentium-233 computer and the capability description of an agent that may provide the price and contact phone number for that computer dealer.

Different users in different situation may want to have different types of matches. Thus, we did provide the matchmaker agent with several kinds of filters.

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 semantical matching filter are meant for plug-in matching [1]. Based on the given notions of matching we did implement 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.

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

4. **Plug-In Matching Mode**. In this mode, the matchmaker performs the signature and semantical matching only.

Users may select any combinations of these filters according to their demand. For example, when efficiency is the major concern, one may use only the context and profile filter. On the other hand, when a user or agent want to find some agents that can perform the task exactly he wants, then he must use the signature and constraint filters to find the plug-in matches. We will now describe each filter in a more detail.

3.2.1 Context Filter

It is obvious that any matching of two specifications has to be in an appropriate context. In LARKS there are two possibilities to deal with that. First, by comparing words in the `Context` slot of considered specifications.

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

3.2.2 Profile Filter

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

Each specification in LARKS is treated as a document, and a word w in a document Req is weighted for that document in the following way. The number of times w occurs throughout all documents is called the document frequency $df(w)$ of w. The used collection of documents is not unlimited, such as the advertisement database of the matchmaker.

Thus, for a given document d, the relevance of d based on a word w is proportional to the number $wf(w,d)$ of times the word w occurs in d and inverse 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:

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

The weighted keyword representation $wkv(d,V)$ of a document d contains for every word w in a given dictionary V the weight $h(w,d)$ as an element. 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 $dps(Req, Ad)$ does exceed a given threshold $\beta \in \mathbf{R}$ the matching process continues with the following steps.

3.2.3 Similarity Filter

The profile filter has two drawbacks: First, it does not consider the structure of the description. That means, e.g., the filter is not able to differentiate among input and output declaration of a specification. Second, comparison of profile does not rely on any semantics of words in a document. Thus, the filter is

not able to recognize that, e.g., the word pair (Computer, Notebook) should have a closer distance than the pair (Computer, Book).

The similarity filter overcomes these drawbacks by comparing descriptions in terms of so-called similarity distance between two descriptions in LARKS. 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 in turn is computed in terms of the distance between concepts and words which occur in the descriptions. The values are computed offline and stored in the respective databases of the matchmaker. Word distance is computed using the trigger-pair model [15]. If two words are significantly co-related, then they are considered as so-called trigger-pairs. The value of the co-relation is domain specific. In the current implementation we use the Wall Street Journal corpus of 1M words 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. Both filters are designed to look for so-called semantical plug-in matches.

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 the concept subsumption testing (see [17] for details).

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

- Their signatures matches.
- 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. This type of semantical match is shown in figure 2.

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

3.3 Concept Subsumption Checking

Concept subsumption relation and concept distance are frequently used in the matching engine, especially in the similarity filter, the signature filter, and the constraint filter. These relations are computed offline and stored in the Concept Database.

A concept C subsumes another concept C' if the extension of C' is a subset of that 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 it is for concept subsumption among two concepts defined in that language. The concept language ITL we use is NP-complete decidable. We use an incomplete inference algorithm for computing subsumption relations among concepts in ITL [2]. For the mechanism of subsumption computation we refer the reader to, e.g., [19].

[2] The well-known trade-off between expressiveness and tractability of concept languages in practice is surrounded almost by subsumption algorithms which are correct but incomplete.

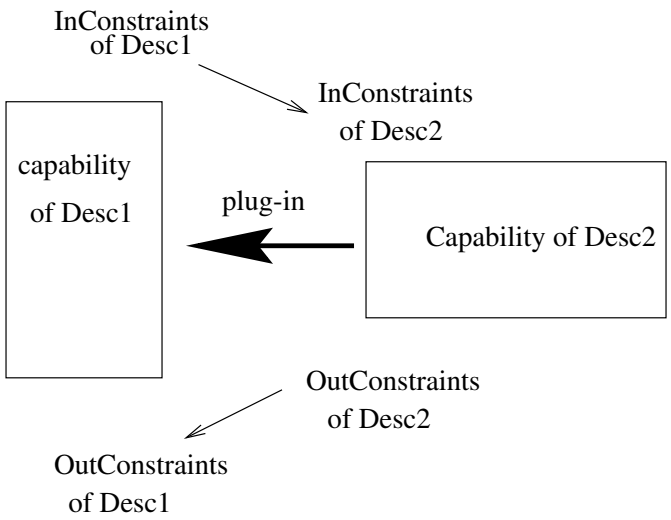

Figure 2: Plug-in match of descriptions: Desc2 plugs into Desc1.

3.4 Computation of Distances Among Concepts

For matchmaking the identification of additional relations among concepts other than subsumption is very useful because it leads to a deeper semantic understanding. Moreover, since the expressivity of the concept language ITL is restrictive so that performance can be enhanced, we need some way to express additional associations among concepts.

For this purpose we use a so-called weighted associative network (AN), that is a semantic network with directed edges between concepts as nodes. Any edge denotes the kind of a binary relation among two concepts and is labeled in addition 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 create an associative network by using the concept subsumption hierarchy and additional associations set by the user. Distance among two concepts C, C'in an AN is computed as the strength of the shortest path between C and C' based on triangular norms (see [3] for details).

4 Example of Matchmaking Using LARKS

Consider the following simple specifications 'IntegerSort' and 'GenericSort' as a request of sorting integer numbers and an advertisement for some agent's capability of sorting real numbers and strings, respectively.

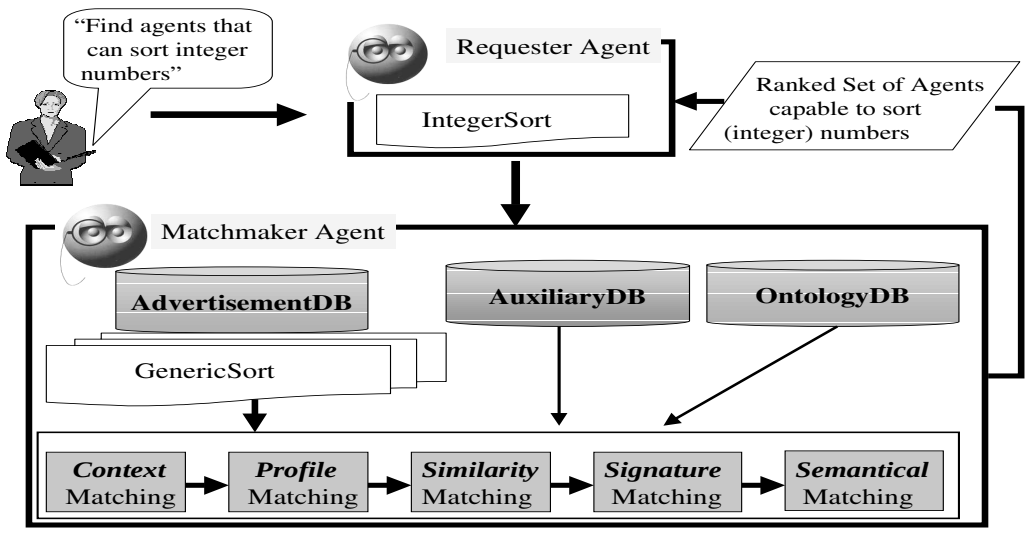

Figure 3: An Example of Matchmaking using LARKS.

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 requester and provider agent sends the request IntegerSort and advertisment GenericSort to the matchmaker, respectively. Figure 4 describes the overall matchmaking process for that request.

1. *Context Matching*
   Both words in the `Context` declaration parts are sufficiently similar. We have no referenced concepts to check for terminologically equity. Thus, the matching process proceeds with the following two filtering stages.
2. *Comparison of Profiles*
   According to the result of TF-IDF method both specifications are sufficiently similar:

3. *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

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

4. *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 $fsm(D_{11}, D_{21}) =$ sub. Analogous for $fsm(D_{12}, D_{22}) =$ sub.

5. *Constraint Matching*
   The advertisement `GenericSort` also plug-in matches semantically with the request `IntegerSort`, because the set of input constraints of `IntegerSort` is θ-subsumed by that of `GenericSort`, and the set of output constraints of `GenericSort` is θ-subsumed by that of `IntegerSort`. Thus `GenericSort` plugs into `IntegerSort`. Please note that this does not hold vice versa.

5 Related Works

Agent matchmaking has been actively studied since the inception of software agent research. The earlist 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 the Horn clauses. The matching between the advertisement and request expressed in KIF is the simple unification with the equality predicate. Matchmaking using LARKS performs better than ABSI in both, the language and the matching process. The plug-in matching in LARKS uses the θ-subsumption test, which select more matches that are also semantically matches.

The SHADE and COINS[10] are matchmakers based on KQML. The content language of COINS allowes for the free text and its matching algorithm utilizes the tf-idf. The contect language of SHADE matchmaker consists of two parts, one is a subset of KIF, another is a structured logic representation called MAX. MAX use logic frames to declaratively store the knowledge. SHADE uses a frame like representation and the matcher use the prolog like unifier.

A more recent service broker-based information system is InfoSleuth[6]. The content language supported by InfoSleuth is KIF and the deductive database language LDL++, which has a 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 which are relevant to the user request.

A somewhat related research area is the research on information mediators among heterogenous information systems[21, 1]. Each local information system is wrapped by a so-called wrapper agent and their capabilities are described in two levels. One is what they can provide, usually described in the local data model and local database schema. Another is what kind of queries they can answer; usually it is a subset of the SQL language. The set of queries a service can accept is described using a grammar-like notation. The 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 sevices. Those systems are not supposed to be searched for among a vast number of resources on the Internet. 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 problem of capability and service descriptions can be tackled at least from the following different approaches:

1. Software specification techniques.
   Agents are computer programs that have some specific characteristics. There are numerous work for software specifications in formal methods, like model-oriented VDM and Z[13], or algebraic-oriented Larch. Although these languages are good at describing computer programs in a precise way, the specification usually contains too much details to be of interests to other agents. Besides, those existing languages are so complex that the semantic comparison between the specifications is impossible. The reading and writing of these specifications also require 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 the STRIPS. The action representation formalism are inadequate in our task in that they are propositional and not involving 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] is developed as an attempt to describe the information sources on the Internet, such that an automated integration of information is possible. In this approach the information source is modeled as a database with restricted quering capabilities.

5.2 Works Related with Service Retrieval

There are three broad approaches to service retrieval. One is the information retrieval techniques to search for relevant information based on text, another is the software component retrieval techniques[22][5][8] to search for software components based on software specifications. The third one is to search for web resources that are typically described as database models[11][21].

In the software component search techniques, [22] defined several notions of matches, including the exact match and the plug-in match, and formally proved the relationship between those matches. [5] proposed to use a sequence of filters to search for software components, for the purpose to increase the efficiency of the search process. [8] computed the distance between similar specifications. All these work are based on the algebraic specification of computer programs. No concept description and concept hierarchy are considered.

In Web resource search techniques, [11] proposed a method to look for better search engines that may provide more relevant data for the user concerns, and rank those search engines according to their relevance to user's query. They propose the directory of services to record descriptions of each information server, called a server description. A user sends his query to the directory of services, which determins and ranks the servers relevant to the user's request. Both the query and the server are described using boolean expression. 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 others 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 capability to a middle agent that stores the advertisements. An agent that desires a particular service sends a middle

agent a service request that is subsequently matched with the middle agent's 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 of their adevertisement 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 advertisement and request and a matchmaking process using LARKS. LARKS judiciously balances language expressivity and efficiency in matching. LARKS 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, namely context matching, comparison of profiles, similarity matching, signature matching and constraint matching. Different degrees of partial matching can result from utilizing different combinations of these filters. 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.

[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 c

[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.

Matchmaking among Heterogeneous Agents on the Internet*

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

Jianguo Lu
Department of Computer Science, University of Toronto, Canada.
jglu@cs.toronto.edu

Abstract

The Internet is not only providing 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 infrastructure framework.

Keywords: Matchmaking, Agent collaboration and coordination

1 Introduction

Nowadays the Internet is not only providing 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 searching and selection of relevant agents is essential for both users and the agent society in several ways. Firstly, novice users in the syberspace may have no idea where to find the service, what agents are available for doing their job. Secondly, experienced users may not 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 a standardized, meaningful communication among agents to enable them to perform collaborative task execution.

To facilitate the searching and interoperation among agent on the Internet, we proposed the RETSINA multi-agent infrastructure framework[21]. In this framework, we distinguish 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 others are called *middle agents*.

We have identified different types of middle agents in the Internet, such as matchmakers (yellow page services), brokers, billboards, etc. [2], and experimentally evaluated different protocols for interoperation between providers, requesters and various types of middle agents. W e have also developed protocols for distributed matchmaking [8]. The process of finding an appropriate provider through a middle agent is called *matchmaking*. It has the following general form (Figure 1):

- Provider agents advertise their capabilities such as know-how, expertise, and so on, to middle agents.

*This research has been sponsored in part by Office of Naval Research grant N-00014-96-16-1-1222.

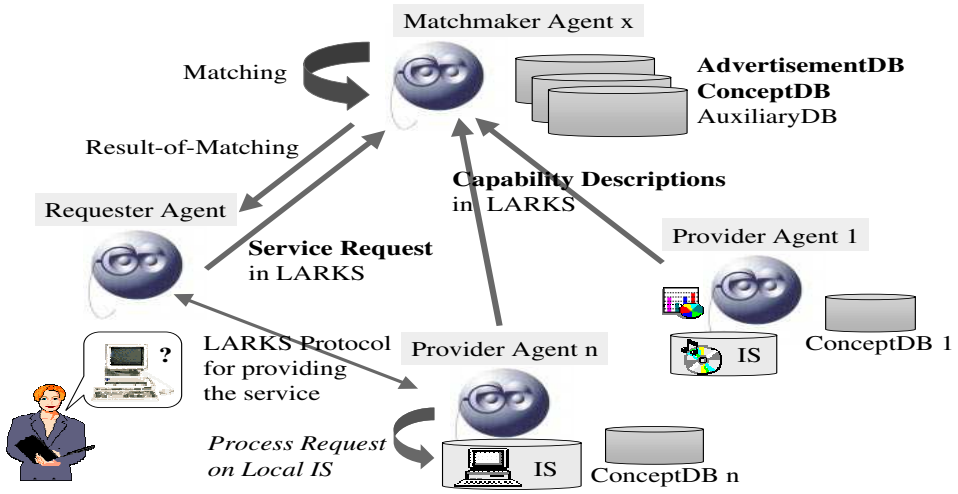

Figure 1: Matchmaking using LARKS: An Overview

- Middle agents store these advertisements.
- 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.

While this process at first glance seems very simple, it is complicated by the fact that providers and requesters are usually heterogeneous and incapable in general 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 [21].

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 of existing agents should be able to be distinguished 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 in the Web. The language allows for automated exchange and processing of information among these agents.

There are many program description languages, Z[15], to describe the functionalities of programs. These languages concern 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[?], WIDL, go to the other extreme by omitting the functional descriptions of the services at all. Only the input and output information are 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 service. Some agent communication languages like KQML[5] and FIPA[?] concentrate on the communication protocals (message types) between agents but leave the content part of the language unspecified.

In Internet computing, various description format are being proposed, notably the WIDL and the Resource Description Framework(RDF)[16]. Although the RDF also aims at the interoperablity between Web applications, it is rather 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 input and 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 declaration, there may have concept attachment. 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 restricted to be 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; this depends on the purpose for which an agent sends a specification to some matchmaker agent(s). Every LARKS specification must be wrapped up in an appropriate KQML message by the sending agent indicating 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 benefit of that option is twofold: (1) the user can specify in more detail what he is requesting or advertising, and (2) the matchmaker agent is able to make automated inferences on such kind of additional semantic descriptions while matching LARKS specifications, thereby improving the overall quality of matching.

Example 2.1: *Finding informations on computers*

Suppose that a provider agent such as, e.g., HotBot, Excite, advertises the capability to find informations about any type of computers. The administrator of the agent may specify that capability in LARKS as follows.

FindComputerInfo	
`Context`	Computer*Computer;
`Types`	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 a given ontology. The definitions of these concepts are included in the slot `ConcDescriptions`. Concept definitions 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[20].

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. But when can we say two descriptions *match* against each other? Does it mean that they have the same text? Or the occurrence of words in one discription sufficiently overlap with those of another discription? When both descriptions are totally different in text, is it still possible for them to match? Even if they match in a given sense, what can we then say about the matched advertisements? Before we go into the details of the matchmaking process, we should clarify the various notions of matches of two specifications.

- **Exact Match** The most accurate match is when both descriptions are equivalent, either equal literally, or equal by renaming the variables, or equal logically obtained by logical inference. This type of matching is the most restrictive one.
- **Plug-In Match** A less accurate but more useful match is the so-called $plug-in$ match. Roughly speaking, plug-in matching means that the agent which capability description matches a given request can be "plugged into the place" where that request was raised. As we can see, exact match is a special case of plug-in match, i.e., wherever two descriptions are exact match, they are also plug-in match.

  A simple example of a plug-in match is that of the match between a request to sort a list of integers and an advertisement of an agent that can sort both list of integers and list of strings. This example is elaborated in section 4. Another example of plug-in match is between the request to find some computer information without any constraint on the output list and the advertisement of an agent that can provide these informations and sorts the respective output.
- **Relaxed Match** The least accurate but most useful match is the so-called *relaxed* match. Relaxed match will not tell whether one description can be reused by another. Instead it determines how close the two descriptions are by returning a numerical distance value. Two descriptions match if the distance value is smaller than a preset threshold value. Normally the plug-in match and the exact match will be a special case of the relaxed match if the threshold value is not too small.

  An example of a relaxed match is that of the request to find the place (or address) where to buy a Compaq Pentium-233 computer and the capability description of an agent that may provide the price and contact phone number for that computer dealer.

Different users in different situation may want to have different types of matches. Thus, we did provide the matchmaker agent with several kinds of filters.

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 semantical matching filter are meant for plug-in matching. Please note, that the computational costs of these filters are in increasing order. Based on the given notions of matching we did implement 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 semantical matching only.

Users may select any combinations of these filters according their demand. For example, when efficiency is the major concern, one may use only the context and profile filter. On the other hand, when a use or agent want to find some agents that can perform the task exactly she wants, then she must use the signature and constraint filters to find the plug-in matches. We will now describe each filter in a more detail.

3.2.1 Context Filter

It is obvious that any matching of two specifications has to be in an appropriate context. In LARKS there are two possibilities to deal with that. First, the `Context` slot in a specification S contains a (list of) words denoting the domain of discourse for matching S with any other specification. When comparing two specifications it is assumed that their domains, means their context, are the same (or atleast sufficiently similar) as long as the real-valued distances between these words do not exceed a given threshold The matching process only proceeds if that is true.

Second, every word in a LARKS specification may be associated with a concept in a given domain ontology. Again, if the context of both specifications turned out to be sufficiently similar in the step before then the concept definitions describe the meaning of the words they are attached to in a more detail in the same domain. In this case, two concepts with same name but different definitions will be stored separately by extending each concept name by the identifier of the agent who did send this concept.

To summarize, the context matching consists of two consecutive steps:

1. For every pair of words u, v given in the `context` slots compute the real-valued word distances $d_w(u,v) \in [0,1]$. Determine the most similar matches for any word u by selecting words v with the minimum distance value $d_w(u,v)$. These distances must not exceed a given threshold.
2. For every pair of most similar matching words, check that the semantic distance among the attached concepts does not exceed a given threshold.

3.2.2 Profile Filter

Although the context matching is efficient, it does not consider the whole specification itself. This is done by the profile filter that compares two LARKS specifications by using the TF-IDF (term frequency-inverse document frequency)[18] technique in information retrieval area.

Each specification in LARKS is treated as a document, and a word w in a document Req is weighted for that document in the following way. The number of times w occurs throughout all documents is called the document frequency $df(w)$ of w. The used collection of documents is not unlimited, such as the advertisement database of the matchmaker.

Thus, for a given document d, the relevance of d based on a word w is proportional to the number $wf(w,d)$ of times the word w occurs in d and inverse 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 for every word w in a given dictionary V the weight $h(w,d)$ as an element. 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 $dps(Req, Ad)$ does exceed a given threshold $\beta \in \mathbf{R}$ the matching process continues with the following steps.

3.2.3 Similarity Filter

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

The similarity filter overcomes these drawbacks by comparing descriptions in terms of so-called similarity distance between two descriptions in LARKS. 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 in turn is computed in terms of the distance between concepts and words which occur in the descriptions. The values are computed offline and stored in the respective databases of the matchmaker. Word distance is computed using the trigger-pair model [17]. If two words are significantly co-related, then they are considered as so-called trigger-pairs. The value of the co-relation is domain specific. In the current implementation we use the Wall Street Journal corpus of 1M words 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 given constraints into account.

The signature filter and the constraint filter are designed to look for the plug-in matches. In our LARKS representation, it is proven that a description Desc2 plug-in matches another description Desc1 if

- Their signatures matches.
- 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[13] relation between clauses. The plug-in semantical match can be illustrated by the following figure.

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

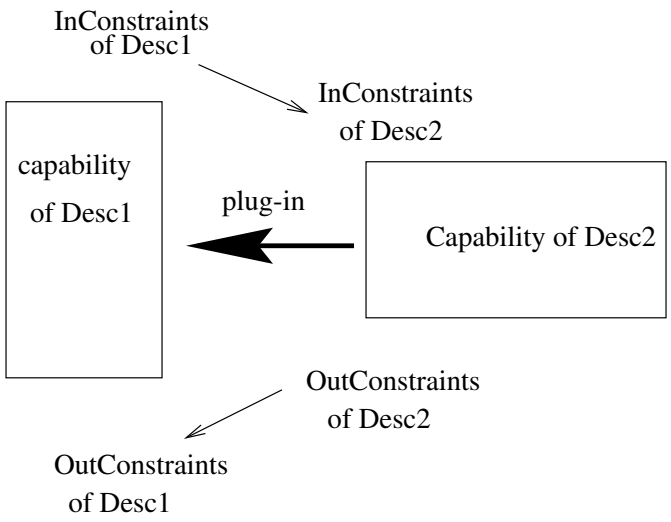

Figure 2: Plug-In Match of Descriptions: T plugs into S.

3.3 Concept Subsumption Checking

Concept subsumption relation and concept distance are frequently used in the matching engine, especially in the similarity filter, the signature filter, and the constraint filter. These relations are computed offline are stored in the Concept Database.

A concept C subsumes another concept C' if the extension of C' is a subset of that 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 it is for concept subsumption among two concepts defined in that language. The concept language ITL we use is NP-complete decidable. The well-known trade-off between expressiveness and tractability of concept languages in practice is surrounded almost by subsumption algorithms which are correct but incomplete. We use an incomplete inference algorithm for computing subsumption relations among concepts in ITL. For the mechanism of subsumption computation we refer the reader to, e.g., [19].

3.4 Computation of Distances Among Concepts

The concept subsumption gives only a generalization/specialization relation based on the definition of the concepts via roles and attribute sets. In particular for matchmaking the identification of additional relations among concepts is very useful because it leads to a deeper semantic understanding. Moreover, since the expressivity of the concept language ITL is restrictive so that performance can be enhanced, we need some way to express additional associations among concepts.

For this purpose we use a so-called weighted associative network (AN), that is a semantic network with directed edges between concepts as nodes. Any edge denotes the kind of a binary relation among two concepts (generalization/specialization, and positive association), and is labeled in addition 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 create an associative network by using the concept subsumption hierarchy and associations set by the user. Distance among two concepts C, C'in an AN is computed as the strength of the shortest path between C and C' based on triangular norms (see [] for details).

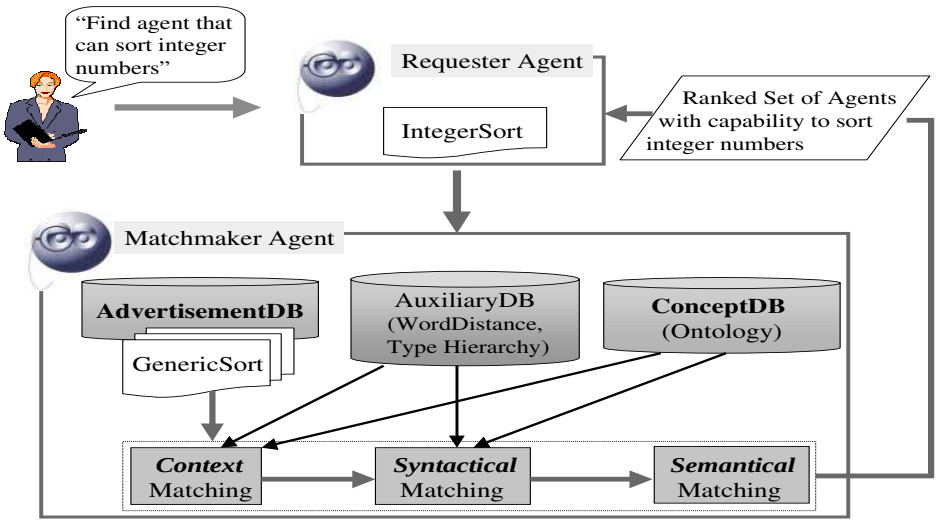

Figure 3: An Example of Matchmaking using LARKS.

4 Example of Matchmaking Using LARKS

Consider the following specifications 'IntegerSort' and 'GenericSort' as a request of sorting integer numbers and an advertisement for some agent's capability of sorting real numbers and strings, respectively.

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 requester and provider agent sends the request IntegerSort and advertisment GenericSort to the matchmaker, respectively. Figure 4 describes the overall matchmaking process for that request.

1. *Context Matching*
   Both words in the `Context` declaration parts are sufficiently similar. We have no referenced concepts

to check for terminologically equity. Thus, the matching process proceeds with the following two filtering stages.

2. *Comparison of Profiles*
   According to the result of TF-IDF method both specifications are sufficiently similar:

3. *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

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

4. *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 $fsm(D_{11}, D_{21}) =$ sub. Analogous for $fsm(D_{12}, D_{22}) =$ sub.

5. *Constraint Matching*
   The advertisement `GenericSort` also plug-in matches semantically with the request `IntegerSort`, because the set of input constraints of `IntegerSort` is θ-subsumed by that of `GenericSort`, and the set of output constraints of `GenericSort` is θ-subsumed by that of `IntegerSort`. Thus `GenericSort` plugs into `IntegerSort`. Please note that this does not hold vice versa.

5 Related Works

Agent matchmaking has been actively studied since the inception of software agent research. The earlist 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 the Horn clauses. The matching between the advertisement and request expressed in KIF is the simple unification with the equality predicate. Matchmaking using LARKS performs better than ABSI in both, the language and the matching process. The plug-in matching in LARKS uses the θ-subsumption test, which select more matches that are also semantically matches.

The SHADE and COINS[11] are matchmakers based on KQML. The content language of COINS allowes for the free text and its matching algorithm utilizes the tf-idf. The contect language of SHADE matchmaker consists of two parts, one is a subset of KIF, another is a structured logic representation called MAX. MAX use logic frames to declaratively store the knowledge. SHADE uses a frame like representation and the matcher use the prolog like unifier.

A more recent service broker-based information system is InfoSleuth[?, 7]. The content language supported by InfoSleuth is KIF and the deductive database language LDL++, which has a 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 which are relevant to the user request.

A somewhat related research area is the research on information mediators among heterogenous information systems[22][1]. Each local information system is wrapped by a so-called wrapper agent and

their capabilities are described in two levels. One is what they can provide, usually described in the local data model and local database schema. Another is what kind of queries they can answer; usually it is a subset of the SQL language. The set of queries a service can accept is described using a grammar-like notation. The 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 sevices. Those systems are not supposed to be searched for among a vast number of resources on the Internet. 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 problem of capability and service descriptions can be tackled at least from the following different approaches:

1. Software specification techniques.
   Agents are computer programs that have some specific characteristics. There are numerous work for software specifications in formal methods, like model-oriented VDM and Z[15], or algebraic-oriented Larch. Although these languages are good at describing computer programs in a precise way, the specification usually contains too much details to be of interests to other agents. Besides, those existing languages are so complex that the semantic comparison between the specifications is impossible. The reading and writing of these specifications also require 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 the STRIPS. The action representation formalism are inadequate in our task in that they are propositional and not involving 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.

   Database query capability description.
   The database query capability description technique is developed as an attempt to describe the information sources on the Internet, such that an automated integration of information is possible. In this approach the information source is modeled as a database with restricted quering capabilities.

5.2 Works Related with Service Retrieval

There are three broad approaches to service retrieval. One is the information retrieval techniques to search for relevant information based on text, another is the software component retrieval techniques[24][6][9] to search for software components based on software specifications. The third one is to search for web resources that are typically described as database models[12][22].

In the software component search techniques, [24] defined several notions of matches, including the exact match and the plug-in match, and formally proved the relationship between those matches. [6] proposed to use a sequence of filters to search for software components, for the purpose to increase the efficiency of the search process. [9] computed the distance between similar specifications. All these work are based on the algebraic specification of computer programs. No concept description and concept hierarchy are considered.

In Web resource search techniques, [12] proposed a method to look for better search engines that may provide more relevant data for the user concerns, and rank those search engines according to their relevance to user's query. They propose the directory of services to record descriptions of each information server, called a server description. A user sends his query to the directory of services, which determins and ranks the servers relevant to the user's request. Both the query and the server are described using

boolean expression. 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 others 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 capability to a middle agent that stores the advertisements. An agent that desires a particular service sends a middle agent a service request that is subsequently matched with the middle agent's 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 of their adevertisement 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 advertisement and request and a matchmaking process using LARKS. LARKS judiciously balances language expressivity and efficiency in matching. LARKS 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, namely context matching, comparison of profiles, similarity matching, signature matching and constraint matching. Different degrees of partial matching can result from utilizing different combinations of these filters. Selection of filters to apply is under the control of the user (or the requester agent).

Acknowledgement: We thank Seth Widoff for invaluable help with the implementation.

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, M. Kracker, E.J. Neuhold. Semantic vs. Structural Resemblance of Classes. Special Issue: Semantic Issues in Multidatabase Systems, ACM SIGMOD RECORD, Vol. 20, No. 4, pp.59-63, 1991.

[4] 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.

[5] 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.

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

[7] 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.

[8] 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.

[9] J-J. Jeng, 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.]

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

[11] 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.

[12] 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.

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

[14] B. Nebel. *Reasoning and revision in hybrid representation systems.* Lecture Notes in Artificial Intelligence LNAI Series, Vol. 422, Springer, 1990.

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

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

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

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

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

[20] 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.

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

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

[23] 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

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