Autonomous Agents and Multi-Agent Systems, 5, 173–203, 2002

LARKS: Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace*

KATIA SYCARA AND SETH WIDOFF katia@cs.cmu.edu
The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA

MATTHIAS KLUSCH klusch@dfki.de
Deduction and Multiagent Systems Lab, DFKI GmbH, Saarbrücken, Germany

JIANGUO LU jglu@cs.toronto.edu
Computer Science Department, University of Toronto, Canada

Abstract. Service matchmaking among heterogeneous software agents in the Internet is usually done dynamically and must be efficient. There is an obvious trade-off between the quality and efficiency of matchmaking on the Internet. We define a language called LARKS for agent advertisements and requests, and present a flexible and efficient matchmaking process that uses LARKS. 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 different 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. We briefly report on our implementation of LARKS and the matchmaking process in Java. Fielded applications of matchmaking using LARKS in several application domains for systems of information agents are ongoing efforts.

Keywords: interoperability, multi-agent systems, matchmaking, capability description

1. Introduction

The amount of services and deployed software agents in the most famous offspring of the Internet, the World Wide Web, is exponentially increasing. In addition, the Internet is an open environment, where information sources, communication links and agents themselves may appear and disappear unpredictably. Thus, an effective, automated search and selection of relevant services or agents is essential for human users and agents as well.

We distinguish three general agent categories in the Cyberspace, *service providers*, *service requester*, and *middle agents*. Service providers provide some type of service, such as finding information, or performing some particular domain specific problem solving. Requester agents need provider agents to perform some service for them. Agents that help locate others are called middle agents [6]. *Matchmaking* is the

* This research has been sponsored in part by Office of Naval Research grant N-00014-96-16-1-1222, and by DARPA grant F-30602-98-2-0138.

process of finding an appropriate provider for a requester through a middle agent, and has the following general form: (1) Provider agents advertise their capabilities to middle agents, (2) middle agents store these advertisements, (3) a requester asks some middle agent whether it knows of providers with desired capabilities, and (4) 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 not only local information sources but even providers and requesters in the Cyberspace are usually heterogeneous and incapable of understanding each other. This gives rise to the need for a common language for describing the capabilities and requests of software agents in a convenient way. Besides, one has to devise an efficient mechanism to determine a structural and semantic match of descriptions in that language. This means in particular using methods for reconciling potentially semantic heterogeneous informations [23]. There is an obvious trade-off between the quality and efficiency of matchmaking on the Internet.

In the following, we briefly present the agent capability description language, LARKS, and then discuss the matchmaking process using LARKS. The paper concludes with a brief comparison 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 [44].

2. Matchmaking among heterogeneous agents

In the process of matchmaking (see Figure 1) three different kinds of collaborating agents involved are:

1. *Provider* agents provide their capabilities, e.g., information search services, retail electronic commerce for special products, etc., to their users and other agents.

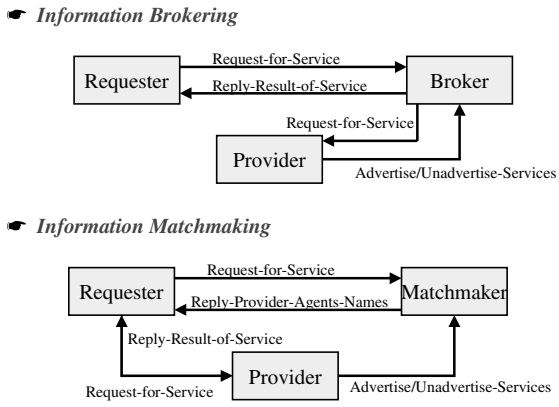

Figure 1. Service brokering vs. matchmaking.

2. *Requester* agents consume informations and services offered by provider agents in the system. Requests for any provider agent capabilities have to be sent to a matchmaker agent.
3. *Matchmaker* agents mediate among both, requesters and providers, for some mutually beneficial cooperation. Each provider must first register himself with a matchmaker. Provider agents advertise their capabilities (advertisements) by sending some appropriate messages describing the kind of service they offer.
   Every request a matchmaker receives will be matched with his actual set of advertisements. If the match is successful the matchmaker returns a ranked set of appropriate provider agents and the relevant advertisements to the requester.

In contrast to a broker agent, a matchmaker does not deal with the task of contacting the relevant providers, transmitting the service request to the service provider and communicating the results to the requester. This avoids data transmission bottlenecks, but it might increase the amount of interactions among agents.

2.1. Agent capability description language requirements

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 major problems in the area of software engineering and AI. Some of the main desired features of such a agent capability description language are the following.

- *Expressiveness*: The language is expressive enough to represent not only data and knowledge, but also to describe the meaning of program code. Agent capabilities are described at an abstract rather than implementation level.
- *Inferences*: Inferences on descriptions written in this language are supported. A user can read any statement in the language, and software agents are able to process, especially to compare any pair of statements automatically.
- *Ease of Use*: Every description 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.

In addition, the matchmatching process on a given set of capability descriptions and a request, both written in the chosen ACDL, should be efficient, accurate—not only relying on keyword extraction and comparison—and fully automated.

3. The agent capability description language LARKS

Representing capabilities is a difficult problem that has been one of the major concerns in the areas of software engineering, AI, and more recently, in the area of

Internet computing. There are many program description languages, like the Vienna Development Method (VDM), VDM++ [29] or Z [35], to describe the program functionality. These languages concern too detail-rich to be feasibly searched. Also, reading and writing specifications in these languages require sophisticated training. On the other hand, the interface definition languages, like WIDL [47], go to the other extreme by omitting the functional descriptions of the services entirely. Only the input and output signature information are provided.

In AI, knowledge description languages, like KL-ONE [3], or knowledge interchange formats such as KIF [22] 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 [10] and FIPA ACL [11, 12] concentrate on specifying communication performatives (message types) between agents but leave the content part of the language unspecified.

In Internet computing, various description formats are being proposed, notably the Web Interface Definition Language (WIDL) [47] and the Resource Description Framework (RDF) [36]. Although the RDF also aims at the interoperablity between Web applications, it is intended rather to be a basis for describing metadata. RDF allows different vendors to describe the properties and relations between resources on the Web. That enables other programs, like Searchbots, to automatically 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 *services* available in the Web.

Since no existing language satisfies our requirements, we propose an ACDL, called LARKS (Language for Advertisement and Request for Knowledge Sharing) that enables advertising, requesting and matching agent capabilities.

3.1. 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
`TextDescription`	Textual description of specification

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 concepts themselves are 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.[1]
- `ConcDesriptions`: Optional description of the meaning of words used in the specification. The description relies on concepts defined in a given local domain ontology. Attachment 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 `ConcDescriptions`.
- `TextDescription`: Optional text description of the meaning of the specification as a request for or advertisement of agent capabilities. In addition, the meaning of input and output declaration, type and context part of the specification may be described by attaching textual comments.

In our current implementation we assume each local domain ontology to be written in the concept language ITL (Information Terminological Language) [43]. Following section gives examples for how to attach concepts defined in this language in a LARKS specification, and also shows an example domain ontology in ITL. A generic interface for using ontologies in LARKS expressed in languages other than ITL will be implemented in near future.

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.

3.2. Examples of specifications in LARKS

The following two examples show how to describe in LARKS the capability to sort a given list of items, and return the sorted list. Example 3.1 is the specification of the capability to sort a list of at most 100 integer numbers, whereas in Example 3.2 a more generic kind of sorting real numbers or strings is specified in LARKS. Since the `ConcDescriptions` slot is empty, i.e., there is no concept attachment in the specification, the semantics of used words in it are assumed to be known to the matchmaker. Examples of how to use concept attachments in a specification are given in the next section.

Example 3.1 (Sorting integer numbers)

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	
TextDescription	sort list of at most 100 integer numbers

Example 3.2 (Generic sort of real numbers or strings)

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	
TextDescription	*sorting list of real numbers or strings*

The next example is a specification of an agent's capability to buy stocks from particular companies, e.g., IBM, Apple or HP, at a stock market.

Example 3.3 (Selling stocks by a portfolio agent)

sellStock	
Context	Stock, StockMarket;
Types	StockSymbols = {IBM, Apple, HP, SIEMENS, Daimler-Chrysler}, Money = Real;
Input	symbol: StockSymbols; yourMoney: Money; shares: Money;
Output	yourStock: StockSymbols; yourShares: Money; yourChange: Money;
InConstraints	yourMoney >= shares*currentPrice(symb);
OutConstraints	yourChange = yourMoney − shares*currentPrice(symb); yourShares = shares; yourStock = symbol;
ConcDescriptions	
TextDescription	buying stocks from IBM, Apple, HP, SIEMENS, or Daimler-Chrysler at the stock market.

Given the name of the stock, the amount of money available for buying stocks and the shares for one stock, the agent is able to order stocks at the stock market. The constraints on the order are that the amount for buying stocks given by the user covers the shares times the current price for one stock. After performing the

order the agent will inform the user about the stock, the shares, and the gained benefit.

3.3. Using domain knowledge in LARKS

As mentioned before, 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. An example for such a domain ontology is given in the next section.

Local ontologies can be formally defined using, for example, concept languages such as ITL, BACK, LOOM, CLASSIC or KRIS, a full-fledged first order predicate logic, such as the knowledge interchange format (KIF) [22], or even the unified modeling language (UML) [13].

The main benefit of using domain knowledge in LARKS specifications is twofold:

1. the user can specify in more detail what she/he is requesting or advertising, and
2. the matchmaker agent is able to make automated inferences on such kind of additional, formally defined semantic descriptions while matching LARKS specifications, thereby improving the overall quality of matching.

As mentioned before, our current implementation of LARKS assumes the domain ontology to be written in the concept language ITL [43]. The research area on concept languages (or description logics) in AI has its origins in the theoretical deficiencies of semantic networks in the late 70's. KL-ONE [3] was the first concept language providing a well-founded semantics for a more natural language-based description of knowledge. Since then different concept languages have been intensively investigated; they are almost all decidable fragments of first-order predicate logic. The following is a simple example for a request and an advertisement written in LARKS in the air combat mission domain.

Example 3.4 (A request and advertisement of agent capabilities). We applied the matchmaking process using LARKS in the application domain of air combat missions. As an example for specification consider the following request and advertisement, 'ReqAirMissions' and 'AWAC-AirMissions,' respectively. The request is to find an agent which is capable to give information on deployed air combat missions launched in a given time interval. Some provider agent in this domain advertises his capability to provide information about a special kind of (AWAC) air combat missions.

ReqAirMissions	
`Context`	Attack, Mission*AirMission
`Types`	Date = (mm: Int, dd: Int, yy: Int), DeployedMission = ListOf(mType: String, mID:String‖Int)
`Input`	sd: Date, ed: Date

Output	missions: Mission
InConstraints	sd <= ed.
OutConstraints	deployed(mID), launchedAfter(mID,sd), launchedBefore(mID,ed).
ConcDescriptions	AirMission = (and Mission (atleast 1 has-airplane) (all has-airplane Airplane) (all has-MissionType aset(AWAC,CAP,DCA,HVAA)))
TextDescription	capable of providing information on deployed air combat missions launched in a given time interval

AWAC-AirMissions

Context	Combat, Mission*AWAC-AirMission
Types	Date = (mm: Int, dd: Int, yy: Int) DeployedMission = ListOf(mt: String, mid:String‖Int, mStart: Date, mEnd: Date)
Input	start: Date, end: Date
Output	missions: DeployedMission;
InConstraints	start <= end.
OutConstraints	deployed(mID), mt = AWAC, launchedAfter(mid,mStart), launchedBefore(mID,mEnd).
ConcDescriptions	AWAC-AirMission = (and AirMission (atleast 1 has-airplane) (atmost 1 has-airplane) (all has-airplane aset(E-2)))
TextDescription	capable of providing information on deployed AWAC air combat missions launched in some given time interval

Suppose that a provider agent such as, for example, HotBot, Excite, or even a meta-searchbot, like SavvySearch or MetaCrawler, advertises the capability to find informations about any type of computers. The administrator of the agent may specify that capability in LARKS as follows.

Example 3.5 (Finding informations on computers)

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]

Please note that provider and requester agents do not have to share the meaning of any words used in LARKS specifications. For example, suppose that the agents do not share the meaning of the word 'Computer' listed as a keyword in the `Context` slot of both, an advertisement and request, respectively. Without any concept attachment the matchmaker agent matches both specifications to be in the same context though they may refer to different domains of discourse.

Any knowledge on relations among concepts attached to a pair of words to be compared when matching two specifications helps the matchmaker agent to determine the semantic similarity between these words. All attached concepts in a given specification are formally defined in a local domain ontology[2] of provider or requester agent.

When multiple domain ontologies exist the matchmaker agent has to cope with the known ontological mismatch problem. If the agents share a common domain ontology equal or different names of concepts possess the same or different semantics, respectively. However, the more difficult case occurs when the agents do not share the same domain ontology; this may occur, for example, when agent capabilities were specified in the same application domain by different people. In this case, equality of concept names does not necessarily mean the equality of their semantics but has to be determined by the matchmaker agent using the concept definitions.[3] For this purpose the matchmaker agent dynamically builds and maintains a partially global terminology based on the received concept definitions. It is assumed that the vocabulary of basic words used in the definition of concepts of this terminology is dynamically shared by the providers and requesters. This provides a minimal common basis for a well-founded canonical interpretation of any concept in the ontology of the matchmaker.

3.3.1. Example for a domain ontology in the concept language ITL. Conceptual knowledge about a given application domain, or even common-sense, may be defined by a set of concepts and roles as terms in a given concept language. In the current implementation of LARKS we use the concept language ITL for this purpose. Each term as a definition of some concept C is a conjunction of logical constraints which

are necessary for any object to be an instance of C. The set of terminological definitions forms a particular style of an ontology, the *terminology*. Any definition of concepts in a terminology relies on

- a set of concepts and roles already defined in the terminology and/or
- a given basic vocabulary of words (primitive components) which are not defined in the terminology, that is, their semantics are assumed to be known and consistently used across boundaries.

The following terminology, is written in the concept language ITL and defines concepts in the computer application domain. It may be used in Example 3.5 in the former section.

```
Product       = (and (all is-manufactured-by Brand) (atleast 1 is-manufactured-by)
                  (all has-price Price))
Computer      = (and Product (exists has-processor CPU) (all has-memory Memory)
                  (all is-model ComputerModel))
Notebook      = (and Computer (all has-price
                  (and (and (ge 1000) (le 2999)) (all in-currency aset(USD)))
                  (all has-weight (and kg (le 5)) (all is-manufactured-by Company))
                  (all is-model aset(Thinkpad380, Thinkpad770,Satellite315))))
Brand         = (and Company (all is-located-in State))
State         = (and (all part-of Country) aset(VA,PA,TX,OH,NY))
Company       = aset(IBM,Toshiba,HP,Apple,DEC,Dell,Gateway)
Colors        = aset(Blue,Green,Yellow,Red)
Money         = (and Real (all in-currency aset(USD,DM,FF,Y,P)))
Price         = Money
LowPrice      = (and Price (le 1800) (exists in-currency aset(USD)))
HighPrice     = (and Price (ge 5000) (exists in-currency aset(USD)))
ComputerModel = aset(HP-Vectra,PowerPC-G3,Thinkpad-380, Thinkpad-770,Satellite-315)
CPU           = aset(Pentium,K6,PentiumII,G3,Merced)
```

Obviously, at some point the providers and requesters must share a certain basic vocabulary to enable a meaningful comparison of used concepts. It is assumed that the basic set of primitive words of the partially global terminology of the matchmaker is unique and shared with providers and requesters. The name of the used local terminology or domain ontology is denoted in the KQML message which wraps the LARKS specification.

3.3.2. Subsumption relationships among concepts. One of the main inferences on ontologies written in concept languages is the computation of the *subsumption* relation among two concepts: 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 decidable for concept subsumption between two concepts defined in that language. The concept language ITL, which we use, is NP-complete decidable. We compromise expressiveness of the NP-complete

decidable ITL for (polynomial) tractability in our subsumption algorithm, which is correct but incomplete. For the mechanism of subsumption computation we refer the reader to, for example, [24, 32, 41, 42].

The computation of subsumption relationships between all concepts in a ontology yields a so-called concept hierarchy. Both the subsumption computation and the concept hierarchy are used in the matchmaking process (see Section 4.1.2).

We assume that the subsumption relation between two concepts may be identified with a real world semantic relation. Like in [39], we utilize an injective, domain-independent mapping between primitive components that occur in the concept definitions on the basis of given synonym relations.[4]

The matchmaker computes the subsumption relations between the concepts included in any advertisement he receives from registered provider agents. This yields a (set of) subsumption hierarchies of available concepts from a variety of local domain ontologies. An extension of the partial global ontology of the matchmaker with additional types of relations is presented in Section 4.1.4. Please note, that this ontology is not necessarily the union of all local domain ontologies of providers, and is *dynamically* built by the matchmaker while processing advertisements from registered provider agents. Any user or agent, requester or provider, may browse through the matchmaker's ontology and use the included concepts for describing the meaning of words in a specification of a request or advertisement in LARKS.[5]

4. The matchmaking process using LARKS

As mentioned before, we differentiate between three different kinds of collaborating information agents: provider, requester and matchmaker agents. Figure 2 shows an overview of the matchmaking process using LARKS.

The matchmaker agent processes a received request in the following main steps:

- Compare the request with all advertisements in the advertisement database.
- Determine the provider agents whose capabilities match best with the request. Every pair of request and advertisement has to go through several different filters during the matchmaking process.
- Inform the requesting agent by sending them the contact addresses and related capability descriptions of the relevant provider agents.

For being able to perform a steady, just-in-time matchmaking process the information model of the matchmaker agent is comprised of the following components.

1. *Advertisement database* (ADB). This database contains all advertisements written in LARKS the matchmaker receives from provider agents.
2. *Partial global ontology*. The ontology of the matchmaker consists of all ontological descriptions of words in advertisements stored in the ADB. Such a description is included in the slot `ConcDescriptions` and sent to the matchmaker with any advertisement.

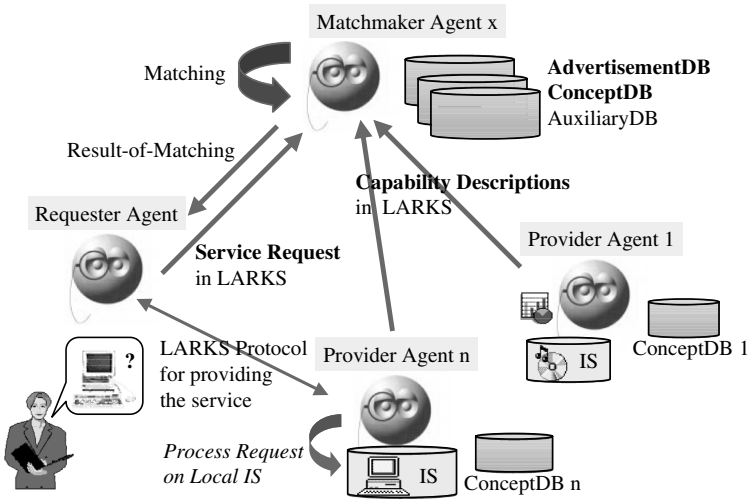

Figure 2. Matchmaking using LARKS: An overview.

3. *Auxiliary database*. The auxiliary data for the matchmaker comprises a database for word pairs and word distances, basic type hierarchy, and internal data.

As mentioned before, the ontology of a matchmaker agent is not necessarily equal to the union of local domain ontologies of all provider agents who are actually registered at the matchmaker. This also holds for the advertisement database. Thus, a matchmaker agent has only partial global knowledge on available information in the overall multi-agent system; this partial knowledge might also be not up-to-date concerning the actual time of processing incoming requests. This is due to the fact that for efficiency reasons changes in the local ontology of a provider agent will not be propagated immediately to all matchmaker agents he is registered at. In the following we will describe the matchmaking process using LARKS in more detail.

4.1. Filtering stages of the matchmaking process

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 description sufficiently overlap with those of another description? 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 types of matches of two specifications.

4.1.1. Types of matching in LARKS

4.1.1.1. Exact match. Of course, 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.

4.1.1.2. 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 whose capability description matches a given request can be "plugged into the place" where that request was raised. Any pair of request and advertisement can differ in the signatures of their input/output declarations, the number of constraints, and the constraints themselves. As we can see, exact match is a special case of plug-in match, that is, 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 5. Another example of plug-in match is between the request to find some computer information without any constraint on the output and the advertisement of an agent that can provide these informations and sorts the respective output.

4.1.1.3. Relaxed match. The least accurate but most useful match is the so-called *relaxed* match. A relaxed match has a much weaker semantic interpretation than a exact match and plug-in match. In fact, relaxed match will not tell whether two descriptions semantically match or not. Instead it determines how close the two descriptions are by returning just 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 Pentium233 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. Although people usually may prefer to have plug-in matches, such a kind of match does not exist in many cases. Thus, people may try to see the result of a relaxed match first. If there is a sufficient number of relaxed matches returned a refined search may be performed to locate plug-in matching advertisements. Even when people are interested in a plug-in match for their requests only, the computational costs for this type of matching might outweigh its benefits.

4.1.2. Different filters of matching in LARKS. For the matchmaking process we adopt several different methods from the area of information retrieval, AI and software engineering for computing syntactical and semantic similarity among agent capability descriptions. These methods are particularly efficient in terms of

performance as needed for dynamic matchmaking in the Internet. To summarize, the matching process is designed with respect to the following criteria:

- The matching should *not be based on keyword retrieval only*. Instead, unlike the usual free text search engines, the semantics of requests and advertisements should be taken into consideration.
- The matching process should be *automated*. A vast amount of agents appear and disappear in the Internet. It is nearly impossible for a user to manually search or browse all agents capabilities.
- The matching process should be *accurate*. For example, if the matches returned by the match engine are claimed to be exact match or plug-in match, those matches should satisfy the definitions of exact matching and plug-in matching.
- The matching process should be *efficient*, that is, it should be fast.
- The matching process should be *effective*, that is, the set of matches should not be too large. For the user, typing in a request and receiving hundreds of matches is not necessarily very useful. Instead, we prefer a small set of highly rated matches to a given request.

To fulfill the matching criteria listed above, the matching process is organized as a series of five increasingly stringent filters on candidate agents:

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

All filters are independent from each other; each of them narrows the set of matching candidates with respect to a given filter criterion. The computational costs of these filters are in increasing order. Users may select any combination of these filters on demand. For example, when efficiency is the major concern, a user might select only the context and profile filters (similar to most conventional SearchBots in the Internet).

Context matching selects those advertisements in the ADB which can be compared with the request in the same or similar context. This filter roughly prunes off advertisements which are not relevant for a given request. The comparison of profiles, similarity and signature matching compare the request with any advertisement selected by the context matching. The request and advertisement profile comparison uses a weighted keyword representation for the specifications and a given term frequency based similarity measure [38]. The last filter, constraint matching, focus on the (input/output) constraints and declaration parts of the specifications. It checks if the input/output constraints of any pair of request and advertisement logically match (see Section 4.1.5).

Concerning the different types of matching there is the following relation to the different filters used in our matchmaker. The first three filters are meant for relaxed matching, and the signature and constraint matching filter are meant for plug-in matching.

4.1.3. Different matching modes of the matchmaker. Based on the given types and filters of matching we did implement four different modes of matching for the matchmaker:

1. *Complete Matching Mode*. In this mode all filters are considered for matching requests and advertisements in LARKS.
2. *Relaxed Matching Mode*. Only the context, profile and similarity filter are considered.
3. *Profile Matching Mode*. Only the context matching and comparison of profiles is done.
4. *Plug-In Matching Mode*. In this mode, the matchmaker performs only the signature and constraint matching.

If the considered advertisement and request contain conceptual attachments, i.e., ontological descriptions of used words, then in most of the filters, except for the comparison of profiles, we need a way to determine the semantic distance between the defined concepts. For that we use the computation of subsumption relationships and a weighted associative network.

4.1.4. Computation of semantic distances among concepts. We have presented the notion of concept subsumption in Section 3.3.2. But 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, 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.[6] We assume that the semantic network consists of three kinds of binary, weighted relationships: (1) generalization, (2) specialization (as inverse of generalization), and (3) positive association among concepts [7]. The *positive association* is the most general relationship among concepts in the network indicating them as synonyms in some context. Such a semantic network is called an *associative network* (AN).

In our implementation an AN is created by the matchmaker by using the computed concept subsumption hierarchy and additional associations extracted from the WordNet ontology [9]. We assume that the terminological subsumption relation among two concepts in the partial global ontology of the matchmaker may be identified with a real world semantical relation among them. That means, all subsumption relations are used for setting the generalization and specialization relations among concepts in the corresponding AN. Positive association, generalization and specialization relations are transitive.

Table 1. Kind of paths in an AN

	g	s	p
g	g	p	p
s	p	s	p
p	p	p	p

As mentioned above, every edge in the AN is labeled with a fuzzy weight. These weights are set by the user or automatically by default. The distance between two concepts in an AN is then computed as the strength of the shortest path among them. For performance reasons the matchmaker does not deal with dynamically resolving ambiguities due to potential genericity and polysemy in the AN (see, e.g., [8]). Combining the strength of each relation in a path is done by using the following triangular norms for fuzzy set intersections [27]:

$$\begin{aligned} \tau_1(\alpha, \beta) &= \max\{0, \alpha + \beta - 1\} & n &= -1 \\ \tau_2(\alpha, \beta) &= \alpha \cdot \beta & n &= 0 \\ \tau_3(\alpha, \beta) &= \min\{\alpha, \beta\} & n &= \infty \end{aligned}$$

Since we have three different kinds of relationships among two concepts in an AN the kind and strength of a path among two arbitrary concepts in the network is determined as shown in Tables 1 and 2. For a formal discussion of that issue we refer to the work of [7, 8, 26].

For all $0 \leq \alpha, \beta \leq 1$ holds that $\tau_1(\alpha, \beta) \leq \tau_2(\alpha, \beta) \leq \tau_3(\alpha, \beta)$. Each triangular norm is monotonic, commutative and associative, and can be used as axiomatic sceletons for fuzzy set intersection. We restrict ourselves to a pessimistic, neutral, and optimistic t-norm τ_1, τ_2 and τ_3, respectively.

Since these triangular norms are not mutually associative the strength of a path in an associative network depends on the direction of strength composition. This asymmetry in turn might lead to unintuitive derived results: Consider, e.g., a path consisting of just three relations among four concepts C_1, C_2, C_3, C_4 with $C_1 \Rightarrow_{g,0.6} C_2 \Rightarrow_{g,0.8} C_3 \Rightarrow_{p,0.9} C_4$. It holds that $\tau_2(\tau_3(0.6, 0.8), 0.9) = 0.54$, but the strength of the same path in opposite direction is $\tau_2(\tau_2(0.9, 0.8), 0.6) = 0.43$. According to Fankhauser and Neuhold [8] we can avoid this asymmetry by imposing a precedence relation ($3 > 2 > 1$) for strength combination (see Table 3).

The computation of semantic distances among concepts is used in most of the filters of the matching process. We will now describe each of the filters in detail.

Table 2. Strength of paths in an AN

	g	s	p
g	τ_3	τ_1	τ_2
s	τ_1	τ_3	τ_2
p	τ_2	τ_2	τ_2

Table 3. Computational precedence for the strength of a path

	g	s	p
g	2	3	1
s	1	2	1
p	1	1	3

4.1.5. The filters of the matchmaking process

4.1.5.1. Context matching. 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 keywords meant to describe the semantic domain of the service. When comparing two specifications it is assumed that their context or domains are the same (or at least sufficiently similar) as long as (1) the real-valued distances between the roots of considered words do not exceed a given threshold, and (2) the distance between the attached concepts of the pairs of most similar words does not exceed a threshold.

Word distance is computed using the trigger-pair model [37]. 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.

For example, both specifications 'ReqAirMissions' and 'AWACS-AirMissions' (see Example 3.4) pass the context filter as to be in a sufficiently similar context. The most similar word pairs are (Attack, Combat), (Mission, Mission), and the concept AirMission subsumes the concept AWACS-AirMission.

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.

4.1.5.2. Comparison of profiles. The comparison of two profiles relies on a standard technique from the Information Retrieval area, called term frequency-inverse document frequency weighting (TF-IDF) (see [38]). According to that, any specification in LARKS is treated as a document.

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 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:

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

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:

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

where $Req \cdot Ad$ denotes the inner product of the weighted keyword vectors. If the value $dps(Req, Ad)$ does exceed a given threshold $\beta \in \mathbf{R}$ both documents pass the profile filter. For example, the profiles of both specifications in Example 3.4 are similar with degree 0.65.

The matchmaker then checks if the declarations and constraints of both specifications for a request and advertisement are sufficiently similar. This is done by a pairwise comparison of declarations and constraints in two steps:

1. *Similarity matching* and
2. *Signature matching*

4.1.5.3. Similarity matching. The profile filter has two limitations. 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. Besides, profile comparison does not rely on the semantics of words themselves. 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 relies on 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.

Let E_i, E_j be variable declarations or constraints, and $S(E)$ the set of words in E. The similarity among two expressions E_i and E_j is determined by pairwise computation of word distances as follows:

$$Sim(E_i, E_j) = 1 - \left(\left(\sum_{(u, v) \in S(E_i) \times S(E_j)} d_w(u, v)\right) \Big/ |S(E_i) \times S(E_j)|\right)$$

The similarity value $Sim(S_a, S_b)$ among two specifications S_a and S_b in LARKS is computed as the average of the sum of similarity computations among all pairs of declarations and constraints:

$$Sim(S_a, S_b) = \sum_{(E_i, E_j) \in (D(S_a) \times D(S_b)) U (C(S_a) \times C(S_b))} Sim(E_i, E_j) \Big/ |(D(S_a) \times D(S_b)) U (C(S_a) \times C(S_b))|$$

with $D(S)$ and $C(S)$ denoting the input/output declaration and input/output constraint part of a specification S in LARKS, respectively. Both specifications in Example 3.4 pass the similarity filter with a similarity value of 0.83.

4.1.5.4. Signature matching. The similarity filter takes into consideration the semantics of individual words in the description. However, it does not take the meaning of the logical constraints in a LARKS specification into account. 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 semantic plug-in match known in the software engineering area [16, 20, 50].

The signature filter first considers the declaration parts of the request and the advertisement, and determines pairwise if their signatures of the (input or output) variable types match following the type inference rules given below.

Definition 4.1 (Subtyping Inference Rules). Consider two types t_1 and t_2 as part of an input or output variable declaration part (in the form `Input` $v : t_1$; or `Output` $v : t_2$;) in a LARKS specification.

1. Type t_1 is a subtype of type t_2 (denoted as $t_1 \preceq_{st} t_2$) if this can be deduced by the following subtype inference rules.
2. Two types t_1, t_2 are equal ($t_1 =_{st} t_2$) if $t_1 \preceq_{st} t_2$ and $t_2 \preceq_{st} t_1$ with
   (a) $t_1 =_{st} t_2$ if they are identical $t_1 = t_2$
   (b) $t_1 \mid t_2 =_{st} t_2 \mid t_1$ (commutative)
   (c) $(t_1 \mid t_2) \mid t_3 = t_1 \mid (t_2 \mid t_3)$ (associative)

Subtype Inference Rules:

(1) $t_1 \preceq_{st} t_2$ if t_2 is a type variable

(2) $\dfrac{t_1 =_{st} t_2}{t_1 \preceq_{st} t_2}$

(3) t_1, t_2 are sets, $\dfrac{t_1 \subseteq t_2}{t_1 \preceq_{st} t_2}$

(4) $t_1 \preceq_{st} t_1 \mid t_2$

(5) $t_2 \preceq_{st} t_1 \mid t_2$

(6) $\dfrac{t_1 \preceq_{st} t_2, s_1 \preceq_{st} s_2}{(t_1, s_1) \preceq_{st} (t_2, s_2)}$

$$(7)\ \frac{t_1 \preceq_{st} t_2, s_1 \preceq_{st} s_2}{t_1 \mid s_1 \preceq_{st} t_2 \mid s_2}$$

$$(8)\ \frac{t_1 \preceq_{st} t_2}{\texttt{SetOf}(t_1) \preceq_{st} \texttt{SetOf}(t_2)}$$

$$(9)\ \frac{t_1 \preceq_{st} t_2}{\texttt{ListOf}(t_1) \preceq_{st} \texttt{ListOf}(t_2)}$$

Matching of two signatures sig and sig' is defined by a binary string-valued function fsm on signatures with

$$fsm(sig, sig') = \begin{cases} sub & sig' \preceq_{st} sig \\ Sub & sig \preceq_{st} sig' \\ eq & sig =_{st} sig' \\ disj & else \end{cases}$$

Having described both filters of the syntactical matching we now define the meaning of syntactical matching of two specifications written in LARKS.

Definition 4.2 (Syntactical matching of specifications in LARKS). Consider two specifications S_a and S_b in LARKS with n_k input declarations, m_k output declarations, and v_k constraints $n_k, m_k \in \mathbf{N}, k \in \{a, b\}$, two declarations D_i, D_j, and constraints C_i, C_j in these specifications, and V a given dictionary for the computation of weighted keyword vectors. Let β, γ, θ be real threshold values for profile comparison and similarity matching.

- *Declarations D_i and D_j syntactically match* if they are sufficiently similar:

$$Sim(D_i, D_j) \geq \gamma \wedge fsm(D_i, D_j) \neq disj.$$

- *Constraints C_i and C_j syntactically match* if they are sufficiently similar:

$$Sim(C_i, C_j) \geq \gamma.$$

  If both words in every pair $(u, v) \in S(E_i) \times S(E_j)$ of most similar words are associated with a concept C and C', respectively, then the distance among C and C' in the so-called associative network of the matchmaker must not exceed a given threshold value θ.

The syntactical match of two declarations or constraints is denoted by a boolean predicate *Synt*.

- The *specifications S_a and S_b syntactically match* if
  1. their profiles match, that is, $dps(S_a, S_b) \geq \beta$, and
  2. for each declaration or constraint E_i, $i \in \{1, \ldots n_a\}$ in the declaration or constraint part of S_a there exists a most similar matching declaration or constraint

E_j, $j \in \{1, \ldots, n_b\}$ in the declaration or constraint part of S_b such that

$$Synt(E_i, E_j) \wedge Sim(E_i, E_j) = \max\{Sim(E_i, E_y), y \in \{1, \ldots, n_b\}\}$$

(Analogous for each declaration or constraint in S_b.)

3. for each pair of declarations determined in (2.) the matching of their signatures is of the same type, that is, for each (D_i, D_j) in (2.) it holds that the value $fsm(D_i, D_j)$ is the same, and
4. the similarity value $Sim(S_a, S_b)$ exceeds a given threshold.

4.1.5.5. Constraint matching. By using the syntactical filter many matches might be found in a large agent society. Hence, it is important to use some kind of semantic information (other than optionally attached concepts and the associative network) to narrow the search, and to pin down more precise matches. This is done by the constraint filter.

The most common and natural interpretation for a specification (even for a software program) is using sets of pre- and post-conditions, denoted as Pre_S and $Post_S$, respectively. In a simplified notation, any specification S can be represented by the pair $(Pre_S, Post_S)$.

A software component description D_2 'semantically plug-in matches' another component description D_1 if (1) their signatures match, (2) the set of input constraints of D_1 logically implies that of D_2, and (3) set of output constraints of D_2 logically implies that of D_1. In our implmentation the logical implication among constraints is computed using polynomial θ-subsumption checking for Horn clauses [31].

Definition 4.3 (Constraint-based semantic matching of two specifications). Consider two specifications $S(Pre_S, Post_S)$ and $T(Pre_T, Post_T)$.

The specification T *semantically matches* the specification S if

$$(Pre_S \Rightarrow Pre_T) \wedge (Post_T \Rightarrow Post_S)$$

That means, the set of pre-conditions of S logically implies that of T, and the set of post-conditions of S is logically implied by that of T.

Plug-in matching of LARKS specifications is valuable for selecting advertisements which are not as constrained in the input parameters as the considered request, but will return equal or greater number of more specific output parameters. For example, the advertisement 'AWAC-AirMission' plugs into the request 'ReqAirMissions' in Example 3.4.

The problem in performing the semantical matching is that the logical implication 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 have to decide on the expressiveness of the language used to represent the pre- and post-conditions, and to choose a relation that is weaker than logical implication. The θ-subsumption relation [31] among two constraints C, C' (denoted as $C \preceq_\theta C'$) appears to be a suitable choice for semantical matching, because it is computationally tractable and semantically sound.

Plug-in Semantical Matching in LARKS

It is proven in the software engineering area that if the condition of semantical matching in Definition 4.3 holds, and the signatures of both specifications match, then T can be directly used in the place of S, that is, T plugs in S.

Definition 4.4 (Plug-in semantical matching of two specifications). Given two specifications *Spec*1 and *Spec*2 in LARKS then *Spec*2 **plug-in matches** *Spec*1 if

- The signatures of their variable declaration parts matches (Section 4.1.4.3).
- For every clause $C1$ in the set of input constraints of *Spec*1 there is a clause $C2$ in the set of input constraint of *Spec*2 such that $C1 \preceq_\theta C2$.
- For every clause $C2$ in the set of output constraints of *Spec*2 there is a clause $C1$ in the set of output constraints of *Spec*1 such that $C2 \preceq_\theta C1$.

where $\preceq_\theta$ denotes the θ-subsumption relation between constraints.

θ-Subsumption between constraints. One suitable selection of the language and the relation is the (definite program) clause and the so-called θ-subsumption relation between clauses, respectively [31].[7] In the following we will only consider Horn clauses. A general form of Horn clause is $a_0 \vee (\neg a_1) \vee \cdots \vee (\neg a_n)$, where each $a_i, i \in \{1, \ldots, n\}$ is an atom. This is equivalent to $a_0 \vee \neg(a_1 \wedge \cdots \wedge a_n)$, which in turn is equivalent to $(a_1 \wedge \cdots \wedge a_n) \Rightarrow a_0)$.[8] We adopt the standard notation for that clause as $a_0 \leftarrow a_1, \ldots, a_n$; in PROLOG the same clause is written as a_0: $a_1, \ldots, a_n$.

Examples of definite program clauses are

- *Date.year* > 1995, *sorted(computerInfo)*,
- *before*$(x, y, ys) \leftarrow ge(x, y)$, and
- *scheduleMeeting(group1, group2, interval, meetingDuration, meetTime)* ← *belongs* × *(p1,group1), belongs(p2,group2), subset(meetTime, interval), length(meetTime)* = *meetingDuration, available(p1,meetTime), available(p2,meetTime)*.

We say that a clause C θ-**subsumes** another clause D (denoted as $C \succeq_\theta D$) if there is a substitution θ such that $C\theta \subseteq D$. C and D are θ-equivalent if $C \preceq_\theta D$ and D *preceq*$_\theta C$.

Examples of θ-subsumption between clauses are

- $P(a) \leftarrow Q(a) \preceq_\theta P(X) \leftarrow Q(X)$
- $P(X) \leftarrow Q(X), R(X) \preceq_\theta P(X) \leftarrow Q(X)$.

Since a single clause is not expressive enough, we need to use a set of clauses to express the pre and post conditions (that is, the input and output constraints) of a specification in LARKS. A set of clauses is treated as a conjunction of those clauses.

Subsumption between two set of clauses is defined in terms of the subsumption between single clauses. More specifically, let S and T be such sets of clauses. Then, we define that S θ-subsumes T if every clause in T is θ-subsumed by a clause in S.

There is a complete algorithm to test the θ-subsumption relation, which is in general NP-complete but polynomial in certain cases. On the other hand, θ-subsumption

is a weaker relation than logical implication, that is, from $C \preceq_\theta D$ we can only infer that C logically implies D but not vice versa.[9]

5. Examples of matchmaking using LARKS

Consider the specifications 'IntegerSort' and 'GenericSort' (see Examples 3.1 and 3.2) 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`	
`TextDescription`	sort of list of at most 100 integer numbers

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`	
`TextDescription`	sorting a list of real numbers or strings

Assume that the requester and provider agent sends the request IntegerSort and advertisment GenericSort to the matchmaker, respectively. Figure 3 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. *Syntactical Matching.*
   (a) *Comparison of Profiles.* According to the result of TF-IDF method both specifications are sufficiently similar:
   (b) *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 $fsm(D_{11}, D_{21}) = \text{sub}$. Analogous for $fsm(D_{12}, D_{22}) = \text{sub}$.

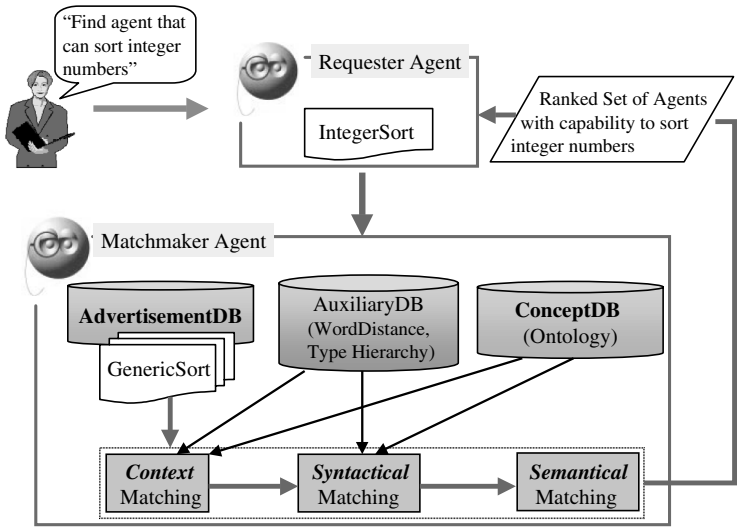

Figure 3. An example of matchmaking using LARKS.

(c) *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:

$$Sim(\texttt{IntegerSort}, \texttt{GenericSort}) = 0.64.$$

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

6. Implementation

We did implement the language LARKS and the matchmaking process using LARKS in Java. Figure 4 shows the user interface of the 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

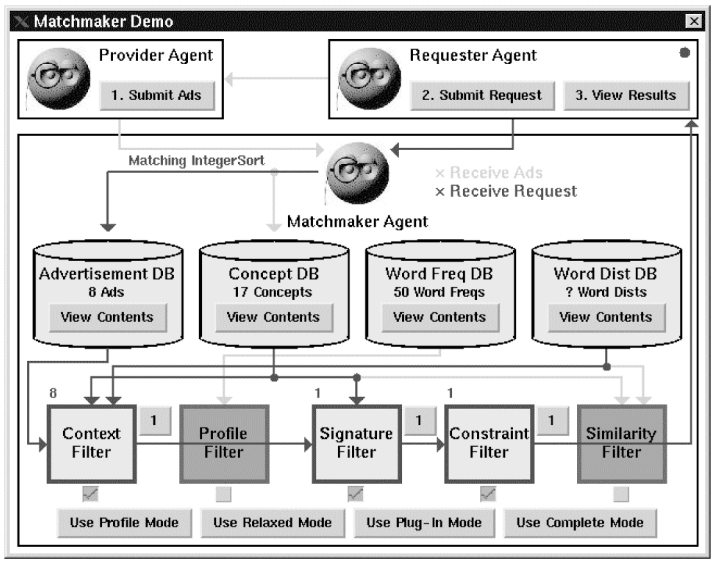

Figure 4. The user interface of the matchmaker agent.

filters—disabled filters are darkened and bypassed. 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.

7. Related work

For dealing with semantic heterogeneity among distributed, autonomous information sources there exist solutions in the multidatabase and information systems area for years. Many of them are based on a database-style modeling of data, global schema, and use of meta-information such as provided by a common ontology or different domain ontologies for a content-based source selection [2, 14, 15, 39]. Others focus on information retrieval (IR) techniques for best-match queries, and relevance assessment. Alternative solutions towards an adaptive process for revealing semantic interdependencies among heterogeneous data objects is proposed, for example, by SCOPES [34].

However, the main problem of dynamic matchmaking in the Internet is to deal with the trade-off between performance and quality of matching. Complex reasoning has to be restricted to allow meaningful semantic matches of requests and advertisements in a reasonable time. Unlike other approaches to matchmaking or brokering in multi-agent systems [2, 28, 33], the presented matchmaking process using LARKS offers a flexible approach to satisfy both requirements. It does not deal with a global integration of heterogeneous source descriptions in terms of database schemas, but

with comparing descriptions of functional capabilities such as constrained actions to provide services. For this purpose it combines techniques from IR, software engineering and description logics area in an appropriate way to perform such filtering efficiently. The matchmaker agent does not need to perform any complex query activities such as, for example, by broker agents in InfoSleuth [33] or the mediator agent in SIMS [2]. In addition, we have developed protocols for efficient, distributed matchmaking among multiple matchmaker agents [19]. We now discuss some of the related works in a more detail.

7.1. Work related to matchmaking and mediation

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 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 [28] are matchmakers based on KQML. The content language of COINS allows 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 uses logic frames to declaratively store the knowledge. SHADE uses a frame like representation and the matcher uses the prolog like unifier.

A more recent service broker-based information system is InfoSleuth [18, 33]. 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.

Another related research area is that on mediators among heterogenous information systems [1, 45]. 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 capabil-

ities and matching are not only studied in the agent community, but also in other related areas.

7.2. *Work related to 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 [35], 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, ontology itself does not describe capabilities. On the other hand, it provides auxiliary concepts to assist the specification of the capabilities of agents.
4. *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.

7.3. *Work related to 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 [16, 20, 50] 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 [30, 45].

In the software component search techniques, Zaremski and Wing [50] defined several notions of matches, including the exact match and the plug-in match, and formally proved the relationship between those matches. Goguen et al. [16] propsed to use a sequence of filters to search for software components, in order to increase the efficiency of the search process. Jeng and Cheng [20] computed the distance between similar specifications. All of these works are based on the algebraic

specification of computer programs. No concept description and concept hierarchy are considered in their work.

In Web resource search techniques, Li and Danzig [30] 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 determines 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.

8. 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 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 matching in 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 semantic 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).

Acknowledgments

We would like to thank Davide Brugali for helpful discussions, and Zhendong Niu and Seth Widoff for help with the implementation of the matchmaker agent using LARKS.

Notes

1. In the future, we plan to allow for using ISO/IEC 13211-1 standard compliant Prolog programs to describe constraints and functional capabilities.
2. For syntax and set-theoretical semantics of used concept language ITL we refer to [43].
3. For methods of determining subsumption or equality of concepts defined in an ontology using a concept language such as ITL we refer to Section 3.3.2.
4. For a further discussion on possible loss of semantics due to mapping among multiple different ontologies we refer to for example [40].
5. This is similar to the common use of domain namespaces in XML [49] for semantically tagging Web page contents.
6. The relationships are fuzzy, and one cannot possibly associate all concepts with each other.
7. A *clause* is a finite set of *literals*, which is treated as the universally quantified disjunction of those *literals*. A *literal* may be positive or negative. A positive *literal* is an *atom*, a negative literal is the negation of an *atom*. A *definite program clause* is a clause with one positive literal and zero or more negative literals. A *definite goal* is a clause without positive literals. A *Horn clause* is either a definite program clause or a definite goal.
8. The literal a_0 is called the head of the clause, and $(a_1 \wedge \cdots \wedge a_n)$ is called the body of the clause.
9. Please also note that the θ-subsumption relation is similar to the query containment in database. When advertisements are database queries, specification matching is reduced to the problem of query containment testing.

References

1. J. L. Ambite and C. A. Knoblock, "Planning by rewriting: Efficiently generating high-quality plans," in *Proc. Fourteenth National Conf. Artif. Intell.*, Providence, RI, 1997.
2. Y. Arens, C. A. Knoblock, and C. Hsu, "Query processing in the SIMS information mediator," in A. Tate (ed.), *Advanced Planning Technology*, AAAI Press: CA, 1996.
3. R. J. Brachman and J. G. Schmolze, "An overview of the KL-ONE knowledge representation system," *Cognitive Science*, vol. 9, no. 2, pp. 171–216, 1985.
4. J. E. Caplan and M. T. Harandi, "A logical framework for software proof reuse," in *Proc. ACM SIGSOFT Symp. Software Reusability, April 1995*. ACM Software Engineering Note, 1995.
5. S. Cranefield, A. Diaz, and M. Purvis, "Planning and matchmaking for the interoperation of information processing agents," The Information Science Discussion Paper Series No. 97/01, University of Otago, 1997.
6. K. Decker, K. Sycara, and M. Williamson, "Middle-agents for the internet," in *Proc. 15th IJCAI*, Nagoya, Japan, August 1997, pp. 578–583.
7. P. Fankhauser, M. Kracker, and 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.
8. P. Fankhauser and E. J. Neuhold, "Knowledge based integration of heterogeneous databases," in *Proc. IFIP Conf. DS-5 Semantics of Interoperable Database Systems*, Lorne, Victoria, Australia, 1992.
9. C. Fellbaum (ed.), *WordNet: An Electronic Lexical Database*, MIT Press, 1998. http://www.cogsci.princeton.edu/wn/
10. T. Finin, R. Fritzson, D. McKay, and R. McEntire, "KQML as an Agent Communication Language," in *Proc. 3rd Int. Conf. Information and Knowledge Management CIKM-94*, ACM Press, 1994.
11. FIPA: Foundation for Intelligent Physical Agents. http://drogo.cselt.it/fipa/, see also, L. Chiariglione, "FIPA—Agent technologies achieve maturity," *AgentLink Newsletter*, vol. 1, November 1999, http://www.agentlink.org
12. FIPA Agent Communication Language. http://www.fipa.org/spec/fipa99spec.htm, 1999.
13. M. Fowler, *UML Distilled: Applying the Standard Object Modeling Language*, Addison-Wesley: Reading, MA, 1997.

14. H. Garcia-Molina, et al., "The TSIMMIS approach to mediation: Data models and languages," in *Proc. Workshop NGITS*, 1995. ftp://db.stanford.edu/pub/garcia/1995/tsimmis-models-languages.ps
15. M. R. Genesereth, A. M. Keller, and O. Duschka, "Infomaster: An information integration system," in *Proc. ACM SIGMOD Conference*, May 1997.
16. J. Goguen, D. Nguyen, J. Meseguer, Luqi, D. Zhang, and V. Berzins, "Software component search," *J. Systems Integration*, vol. 6, pp. 93–134, 1996.
17. G. Huck, P. Fankhauser, K. Aberer, and E. J. Neuhold, "Jedi: Extracting and synthesizing information from the web," in *Proc. Int. Conf. Cooperative Information Systems CoopIS'98*, IEEE Computer Society Press, 1998.
18. N. Jacobs and R. Shea, "Carnot and InfoSleuth—Database technology and the WWW," in *ACM SIGMOD Int. Conf. Management of Data*, May 1995.
19. S. Jha, P. Chalasani, O. Shehory, and K. Sycara, "A formal treatment of distributed matchmaking," in *Proc. Second Int. Conf. Autonomous Agents (Agents 98)*, Minneapolis, MN, May 1998.
20. J-J. Jeng and B. H. C. Cheng, "Specification matching for software reuse: A foundation," in *Proc. ACM SIGSOFT Symposium Software Reusability*, ACM Software Engineering Note, Aug. 1995.
21. V. Kashyap and A. Sheth, "Semantic heterogenity in global information systems: The role of metadata context and ontology," in M. P. Papazoglou and G. Schlageter (eds.), *Cooperative Information Systems: Trends and Directions*, Academic Press, 1998.
22. KIF. Knowledge Interchange Format: http://logic.stanford.edu/kif/
23. W. Kim, et al., "On resolving schematic heterogeneity in multidatabase systems," *Intl. J. on Distributed and Parallel Databases*, vol. 1, pp. 251–279, 1993.
24. M. Klusch, "Cooperative information agents on the Internet," Ph.D. Thesis, University of Kiel, December 1996 (in German) Kovac Verlag, Hamburg, 1998, ISBN 3-86064-746-6.
25. M. Klusch (ed.), *Intelligent Information Agents*, Springer, ISBN 3-540-65112-8, 1999.
26. M. Kracker, "A fuzzy concept network," in *Proc. IEEE Int. Conf. Fuzzy Systems*, 1992.
27. R. Kruse, E. Schwecke, and J. Heinsohn, *Uncertainty and Vagueness in Knowledge Based Systems*, Springer, 1991.
28. D. Kuokka and L. Harrada, "On using KQML for matchmaking," in *Proc. 3rd Int. Conf. on Information and Knowledge Management CIKM-95*, AAAI/MIT Press, 1995, pp. 239–45.
29. K. C. Lano, *Formal Object-oriented Development*. Formal Approaches to Computing and Information Technology Series, Springer-Verlag, 1995.
30. S.-H. Li and P. B. Danzig, "Boolean similarity measures for resource discovery," *IEEE Trans. on Knowledge and Data Engineering*, vol. 9, no. 6, November/December, 1997.
31. S. Muggleton and L. De Raedt, "Inductive logic programming: Theory and methods," *J. of Logic Programming*, vol. 19, no. 20, pp. 629–679, 1994.
32. B. Nebel, *Reasoning and Revision in Hybrid Representation Systems*, Lecture Notes in Artificial Intelligence LNAI Series, vol. 422, Springer, 1990.
33. M. Nodine and J. Fowler, "An overview of active information gathering in Infosleuth," *Proc. Int. Conf. on Autonomous Agents*, USA, 1999, submitted.
34. A. Ouksel, "A framework for a scalable agent architecture of cooperating heterogeneous knowledge sources," in M. Klusch (ed.), *Intelligent Information Agents*, chap. 5, Springer, 1999.
35. B. Potter, J. Sinclair, and D. Till, *Introduction to Formal Specification and Z*, Prentice-Hall International Series in Computer Science, 1996.
36. Resource Description Framework (RDF) Schema Specification. http://www.w3.org/TR/WD-rdf-schema/.
37. R. Rosenfield, "Adaptive statistic language model," Ph.D. thesis, Carnegie Mellon University, 1994.
38. G. Salton, *Automatic Text Processing: The Transformation, Analysis and Retrieval of Information by Computer*, Addison-Wesley, Reading, MA, 1989.
39. A. Sheth, E. Mena, A. Illaramendi, and V. Kashyap, "OBSERVER: An approach for query processing in global information systems based on interoperation across pre-existing ontologies," in *Proc. Int. Conf. on Cooperative Information Systems CoopIS-96*, IEEE Computer Soc. Press, 1996.
40. A. Sheth, A. Illaramendi, V. Kashyap, and E. Mensa, "Managing multiple information sources through ontologies: Relationship between vocabulary heterogeneity and loss of information," in *Proc. ECAI-96*, Budapest, 1996.

41. G. Smolka and B. Nebel, "Representation and reasoning with attributive descriptions," IWBS Report 81, IBM Deutschland Wissenschaftl, Zentrum, 1989.
42. G. Smolka and M. Schmidt-Schauss, "Attributive concept description with complements," *AI* vol. 48, 1991.
43. K. Sycara, J. Lu, and M. Klusch, "Interoperability among heterogeneous software agents on the Internet," Carnegie Mellon University, PA, Technical Report CMU-RI-TR-98-22.
44. K. Sycara, K. Decker, A. Pannu, M. Williamson, and D. Zeng, "Distributed intelligent agents," *IEEE Expert*, pp. 36–46, December 1996.
45. V. Vassalos and Y. Papakonstantinou, "Expressive Capabilities Description Languages and Query Rewriting Algorithms," available at http://www-cse.ucsd.edu/yannis/papers/vpcap2.ps
46. 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
47. WIDL, "The W3C Web Interface Definition Language," http://www.w3.org/TR/NOTE-widl
48. WordNet—A Lexical Database for English. http://www.cogsci.princeton.edu/wn/
49. XML, "Extensible markup language," World Wide Web Consortium (W3C) Working Draft, 17 November 1997. http://www.w3.org/TR/WD-xml-link.
50. A. M. Zaremski and J. M. Wing, "Specification matching of software components," Technical Report CMU-CS-95-127, 1995.