Journal of Web Engineering, Vol.2, No.3 (2004) 131-147

DYNAMIC SERVICE MATCHMAKING IN INTELLIGENT WEB

YUNCHENG JIANG, ZHONGZHI SHI , HAIJUN ZHANG, MINGKAI DONG

Key Laboratory of Intelligent Information Processing, Institute of Computing Technology

The Chinese Academy of Sciences, Beijing 100080, China

jiangyc@ics.ict.ac.cn

Received May 12, 2003
Revised January 28, 2004

Intelligent Web functions essentially as an enormously autonomic entity, and it automatically regulates the functions and cooperation of related Web sites and available application services. Agent is the core component of the Intelligent Web. Intelligent Web not only can present the static information, but also can present dynamic services. In this paper, we study the problems of service management in Intelligent Web and analyze the insufficiencies of the service description language CDL, SDL and LARKS. Combining the features of the Intelligent Web, Web Services, and Grid Services, we propose an agent service description language SDLSIN which satisfies ten properties of agent service description. Based on SDLSIN, we mainly study the service matchmaking problem of Intelligent Web, and propose three kinds of service matchmaking algorithms which are adapted to open and dynamic Intelligent Web. At last, the development of SDLSIN and three kinds of service matchmaking algorithms in Multi-AGent Environment MAGE which we developed is introduced.

Key words: Intelligent Web, Agent Service Description, Dynamic Service Matchmaking
Communicated by: D Schwabe & Q Li

1 Introduction

Intelligent Web functions essentially as an enormously autonomic entity, and it automatically regulates the functions and cooperation of related Web sites and available application services [1]. At present, the Web only presents the static information, but the Intelligent Web not only presents these static information, but also presents dynamic services. Agent is the core component of the Intelligent Web. In Intelligent Web, there are a lot of key issues, such as semantic representation, agent, Web mining, resource management, and service management, need to be resolved. In this paper we will introduce the research work of service management in Intelligent Web. We will use agent technology to realize all kinds of activities which are related to service management, therefore, in this paper we will mainly study how to use agent to manage all kinds of services in Intelligent Web, especially focusing on the agent dynamic service matchmaking in Intelligent Web.

There are many steps in the service management of Intelligent Web, mainly including agent service description, agent service matchmaking, agent service invoke, agent service composition, agent service cooperation, agent service validation, agent service control, and agent service trace and so on. In this paper we will study the first two steps i.e., agent service description and agent service matchmaking in Intelligent Web.

The precondition of realizing service management of Intelligent Web is the realization of agent service description (or capability description) and service matchmaking, but representing capability is a difficult problem that has been one of the major concerns in the area of AI, software engineering, and more recently, in the area of Semantic Web and Intelligent Web.

At present there has been many research work, K Sycara, et al[2] studied and developed the agent capability description language LARKS which considered the trade-off between Quality of Service (QoS) and efficiency of service, but it didn't consider the inheritance mechanism of service description and agent state language. More important it didn't consider the negotiation mechanism of service. G J Wickler[3] studied and developed the agent capability description language CDL which was a action language, but it was a syntax based language and didn't consider the negotiation mechanism of services; on the other hand, CDL didn't support the definition of data types. K Arisha, et al[4] also presented a kind of agent service description language SDL which was a single HTML-like language. This language used hierarchy mechanism and synonym dictionary to study the semantics of service description. We think ontology technology is needed in agent service description. As to all of these agent capability (or service) description languages, there exists corresponded service matchmaking algorithm. In section 6, we will compare these agent capability (or service) description languages and their matchmaking algorithms in detail.

Based on these reasons, we presented a kind of agent service description language SDLSIN(Agent **S**ervice **D**escription **L**anguage with **S**emantics and **I**nheritance and Supporting **N**egotiation)with semantics and inheritance and supporting negotiation. This language not only considered semantic service description of agent, but also considered the inheritance and negotiation mechanism of agent service description, agent state language, and data types, therefore it overcome the insufficiencies of LARKS, CDL, and SDL and improved them. Based on SDLSIN, we mainly studied different dynamic service matchmaking algorithms and strategies using SDLSIN in Intelligent Web.

2 Agent Service Description

Agent service description (or capability description) is an important issue in agent and multi-agent systems research field. As to this problem, G J Wickler[3] gave a summary and presented a kind of agent capability description language CDL. Subsequently K Arisha, et al[4] also presented agent service description language SDL , and K Sycara, et al[2] presented agent capability description language LARKS. In the Introduction of this paper we analyzed their strongpoint and shortcoming respectively.

Combining with the research work of G J Wickler[3], K Arisha[4], K Sycara[2] and J Gonzalez-Castillo[5], we think that agent service description should satisfy the following several properties:

(1) High degree of expressiveness. Agent service description language not only can express information and knowledge, but also can express services and the meaning of program code. Agent service description should be in the abstract level, not be in implemented level. The agent must have total freedom to compose the service description. Different agent will want to describe their services with different degrees of complexity and completeness, and our language must be adaptable to these needs.

(2) High degree of flexibility. Agent service description language may be used by different agents (such as service provider, service requester); at the same time, it may hold inheritance mechanism, agent concept language , and agent state language, etc. On the other hand, an agent service description

may be very descriptive in some points, but leave others less specified and open for negotiation. Therefore, ability to express semi-structured data is required.

(3) Support for inferences. Agent should finish many service management problems autonomously, including service matchmaking, service invoke, service composition, service cooperation and service validation. An agent is able to process, especially to compare any pair of statements automatically, therefore agent service description must possess reasoning capability. In order to support for inferences, agent service description may utilize logic theory, such as description logic and first order logic.

(4) Semantics based service description. Because Intelligent Web is a open, heterogeneous, and dynamic environment, the terminology in each agent service description is often different, if we only use syntax based service description (e.g. keywords based service description), we impossible resolve the interoperation problem of Intelligent Web. Therefore we should use semantics based service description i.e., we need use ontology technology in agent service description.

(5) Support for service negotiation mechanism, because agent is a autonomous entity, and it is different from general Web services. In general Web services , service may be realized through simply invoke the operation, but in Intelligent Web, it utilizes agent to realize service, and agent service invoke, service composition and service cooperation need negotiation between agents.

(6) Support for datatypes. Attibutes such as quantities, prices, or dates will be part of the service description. The best way to express and compare this information is by means of datatypes. As a starting point, we will deal with datatypes such as integer, real, date, string etc.

(7) Adapt to multi-agent systems (MASs). Because Intelligent Web is the development of MASs in Web, according to MASs' characteristics and applications, agent service description should present theory and realization foundation for cooperative solving, coalition formation, and agent coordination of traditional MASs.

(8) Adapt to Web. Because Intelligent Web is the development of Web, the agent service description must be compatible with Web technologies and the information must be in a format appropriate for the Web.

(9) Consider the trade-off between Quality of Service (QoS) and efficiency of service. Because there exists a contradiction between them, some agents regard QoS as the precondition, and some agents regard efficiency of service as the precondition, but some agents need to consider the trade-off between QoS and efficiency of service.

(10) Ease of use. Every description should not only be easy to read and understand, but also easy to write by user. The language should support the use of domain or common ontologies for specifying agents capabilities.

Remark: In this paper, we only study how to describe the agent capability in the Intelligent Web. As to the service invoke and binding method of Intelligent Web, we may reference to the standard of Web Services (e.g. SOAP and WSDL).

3 Agent Service Description Language SDLSIN

According to aforementioned ten properties of agent service description, and aiming at the insufficiencies of CDL, SDL and LARKS, we presented a kind of agent service description language

SDLSIN(Agent **S**ervice **D**escription **L**anguage with **S**emantics and **I**nheritance and Supporting **N**egotiation)with semantics and inheritance and supporting negotiation.

3.1 Component of SDLSIN

SDLSIN is a kind of framework language with slots, and its formal criterion is the following:

```
<asdl-descr>::=(ctype
      :service-name  name
      :context  context-name+
      :types  (type-name = <modifier> type)+
      :isa  name+
      :inputs  (variable: <modifier> put-type-name)+
      :outputs  (variable: <modifier> put-type-name)+
      :input-constraints  (constraint)+
      :output-constrains  (constraint)+
      :io-constrains  (constraint)+
      :concept-description  (ontology-name = ontology-body)+
      :state-language  name
      :concept-language  name
      :attributes  (attributes-name : attributes-value)+
      :text-description  name
    )

ctype ::= capability | task
context-name ::= name<* ontology-name >
type-name ::= name
modifier ::=listof | setof
type ::= (name : name<* ontology-name >)+
put-type-name ::= (type-name | name<* ontology-name >)+
variable ::= name<* ontology-name >
ontology-body ::= (expression in concept-language)
attributes-name ::= name <* ontology-name >
attributes-value ::= name
name ::= String
ontology-name ::= name
constraint ::= (expression in state-language)
```

In this paper and SDLSIN version which we implemented in Java, the agent concept language is description logic[6] (DL), and the agent state language is first order predicate logic (FOPL).

3.2 Meaning of Each Component

In the aforementioned formal criterion of SDLSIN, the meaning of each component is the following:

ctype has two values, one is capability, and the other is task, where capability is the identifier which service provider (SP) registers its capability to the middle agent. And task is the identifier which service requester (SR) requests services from the middle agent.

Remark: In open, heterogeneous, and dynamic Intelligent Web, we distinguish three general agent categories, service provider (SP), service requester (SR), and middle agent (MA). Service providers provide some type of services. Requester agents need provider agents to perform some services for them. Agents that help locate others are called middle agents, and they possess the

capability of service negotiation and service composition, etc. Agent may be a provider, as well as a requester.

service-name denotes the name of service (i.e identifier). In Intelligent Web a service should have unique service-name; at the same time a service-name unique identifies a service. Another purpose of service-name is that it is used the value of attribute **isa** which realized the inheritance relationship between service description.

context uses several keywords to description the main characteristics of service. It mainly used by similarity service matchmaking (syntax based similarity service matchmaking and semantics based similarity service matchmaking).

types denotes the definition of the data types used in the service description. We may define some data types (such as Integer, Real, and String) in advance.

isa allows the name of a service from which this service will inherit the description. It is similar to the OO's inheritance.

inputs denote input variable declarations for the service.

outputs denote output variable declarations for the service i.e., the outcome of invoking service.

input-constraints denote the logical constraints on input variables that appear in the input declaration.

output-constraints denote the logical constraints on output variables that appear in the output declaration.

io-constrains parts define the constraints across input and output variables that must hold. Free variables in these constraints can be from the expressions describing the inputs or outputs.

concept-description denotes the description of the meaning of words used in the service description i.e., in concept-description, some terminologies may be defined in concept-language formally. The description relies on concepts defined in a given local domain ontology.

state-language denotes the constraints language used by attributes input-constraints, output-constraints, and io-constrains. It is often a logic language.

concept-language denotes the description language used by concept-description. It is often a terminology knowledge representation language, such as Description Logic (DL).

attributes mainly support for service negotiation. Its values include cost, quality of service, style of service, and performance of service etc.

text-description is mainly used to descript the service in natural language. Its value mainly is viewed by users.

Except for attribute service-name, all of these attributes are optional. Agent may choose different attributes according to different applications.

From the aforementioned description, it is known that the SDLSIN is a very flexible language, and it is a kind of framework language with slots. Because Intelligent Web is in quick development, and with the development of Intelligent Web, any necessary component may be added to the language in the future, so this SDLSIN language is an open language, and not a complete language.

3.3 Discussion

SDLSIN language satisfies the ten properties of agent service description language.

As to property (1), high degree of expressiveness, and property (2), high degree of flexibility, their proof may be reference to the CDL[3] (in reference [3], the formal definitions of expressiveness and flexibility are defined), because the SDLSIN is the extension of the CDL, and the CDL is expressive and flexible, so the SDLSIN is also expressive and flexible (strictly speaking, SDLSIN is more expressive and flexible than CDL). As to property (3), support for inferences, because the agent state language and agent concept language of SDLSIN is logic language, such as description logic and first order logic, so the property (3) is satisfied. As to property (4), semantics based service description, because SDLSIN utilizes **concept-description** component to describe all terminological words, and these terminological words definitions are the ontologies which used in agent service description, so the property (4) is satisfied. SDLSIN utilizes **attributes** component to describe the negotiation attributes of agent service description, such as service cost, quality of service QoS and service security, etc. so the property (5), support for service negotiation mechanism , is satisfied. SDLSIN utilizes **types** component to define data types which used in agent service description, so the property (6), support for datatypes, is satisfied. Because SDLSIN is a kind of agent service description language, so the property (7), adapt to multi-agent systems (MASs), is satisfied. Because the service description in SDLSIN may be translated into the format in XML which be appropriate for the Web, so the property (8), adapt to Web, is satisfied. Based on the SDLSIN, there exists three kinds of matchmaking algorithms, approximate service matchmaking algorithm, precise service matchmaking algorithm, and plug-in service matchmaking algorithm, if users want high service efficiency, approximate or plug-in service matchmaking algorithms may be used, if users want high service quality, precise service matchmaking algorithm may be used, so the property (9), consider the trade-off between Quality of Service (QoS) and efficiency of service, is satisfied. Because the SDLSIN is a kind of framework language with slots, so the property (8), ease of use, is satisfied.

4 Dynamic Service Matchmaking

4.1 Service Matchmaking Process

Middle Agent has many species[7] [8], such as matchmaker, broker, facilitator, and mediator. These middle agents possess same main functions which help requester locate others agents that need by them.

Service Matchmaking process of finding an appropriate provider for a requester through a middle agent and has the following general form: (1) Provider agents advertise their capability to middle agents; (2) Middle agents store these advertisements and form global terminology ontology; (3) Requester asks the middle agent whether it knows of providers with desired capability; 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 Intelligent Web is a open, heterogeneous, and dynamic system, the information, resources, and services in Intelligent Web are usually incapable of understanding each other. Owing to these reasons, there are main problems in agent service matchmaking of Intelligent Web:

(1) Agent service (or capability) description language.

(2) Agent service matchmaking interaction protocol and agent communication language. We will study this problem in other papers.

(3) Agent service matchmaking algorithm. This algorithm decide which SP's capability satisfies the demand of SR, and it should possess high flexibility: not only support for syntax based service matchmaking, but also support for semantics based service matchmaking; not only consider the quality of service, but also consider the efficiency of service; not only provide precise service matchmaking algorithm, but also provide similar service matchmaking algorithm.

4.2 Service Matchmaking Algorithm

4.2.1 Service Matchmaking Type

When can we say that the service M provided by SP agent can match the service N requested by a SR agent? Before introduce this problem, we will at first introduce the service matchmaking type simply.

(1) Approximate match. This algorithm only demands the M and N is similar, and this similarity either is syntax based similarity, or semantic based similarity.

(2) Exact match. This algorithm demands M and N is semantic equality.

(3) Plug-in match. This algorithm lines in middle between approximate match and exact match, and it demands M subsume N i.e., as to N, M can provide more service, and N can be "plugged in" M.

In fact, exact match is a kind of plug-in match, and plug-in match is a kind of approximate match.

4.2.2 Service Matchmaking Algorithm

4.2.2.1 Preliminaries

Definition 1(SDLSIN Description) a SDLSIN description S is a tuple $S = < A^S, SL^S, CL^S, SN^S, CON^S, TY^S, SUP^S, I^S, O^S, S_I^S, S_O^S, S_{IO}^S, CD^S, ATT^S, TD^S >$, where : A^S is the agent of providing service S; SL^S is the state language of service description S; CL^S is the concept language of service description S; SN^S is the name of service description S; $CON^S = \{con_1^S, ..., con_p^S\}$ is the key characteristics set of service description S; $TY^S = \{ty_1^S, ..., ty_q^S\}$ is the data type definition set of service description S; $SUP^S = \{\sup_1^S, ..., \sup_r^S\}$ is parent class type set of service description S; $I^S = \{I_1^S, ..., I_{i_s}^S\}$ is the input parameters declaration set of service description S; $O^S = \{O_1^S, ..., O_{j_s}^S\}$ is the output parameters declaration set of service description S; $S_I^S = \{S_{I1}^S, ..., S_{Ik_s}^S\}$ is the input constraints set of service description S; $S_O^S = \{S_{O1}^S, ..., S_{Ol_s}^S\}$ is output constraints set of service description S; $S_{IO}^S = \{S_{IO1}^S, ..., S_{IOm_s}^S\}$ is input-output constraints set of service description S; $CD^S = \{cd_1^S, ..., cd_t^S\}$ is the terminologies set used by service description S; $ATT^S = \{att_1^S, ..., att_n^S\}$ is the negotiation attributes set of service description S; TD^S is the natural language description of service description S.

Remark: Definition 1 is the formalization of service description language SDLSIN. In this paper, in order to simplify, we will simplify S as $< A^S, SN^S, CON^S, TY^S, SUP^S, I^S, O^S,$

$S_I^S, S_O^S, S_{IO}^S, CD^S >$ i.e., we assume agent state language is FOPL; agent concept language is DL. As to negotiation attribute ATT and natural language description TD, service matchmaking algorithm needn't them.

Because there may exist attribute **isa** in SDLSIN service description i.e., a service may inherit another (or some) service(or services), so if service description PS has **isa** attribute, we firstly need instantiate PS, then to match service.

Definition 2(Service Instantiation Condition) Given services description S_1,…, S_i and S_k, they are respectively:

$< A^{S_1}, SN^{S_1}, CON^{S_1}, TY^{S_1}, SUP^{S_1}, I^{S_1}, O^{S_1}, S_I^{S_1}, S_O^{S_1}, S_{IO}^{S_1}, CD^{S_1} >$

$\vdots$

$< A^{S_i}, SN^{S_i}, CON^{S_i}, TY^{S_i}, SUP^{S_i}, I^{S_i}, O^{S_i}, S_I^{S_i}, S_O^{S_i}, S_{IO}^{S_i}, CD^{S_i} >$

$< A^{S_k}, SN^{S_k}, CON^{S_k}, TY^{S_k}, SUP^{S_k}, I^{S_k}, O^{S_k}, S_O^{S_k}, S_O^{S_k}, S_{IO}^{S_k}, CD^{S_k} >$, if S_k can be instantiated by S_1,…, S_i, then the following condition must be satisfied:

(1) $SL^{S_1} = ... = SL^{S_i} = SL^{S_k}$;

(2) $CL^{S_1} = ... = CL^{S_i} = CL^{S_k}$;

(3) $SUP^{S_k} = SN^{S_1} \bigcup ... \bigcup SN^{S_i}$.

Definition 3(Attribute-Unifying Substitution) Given services description S_1 and S_2, assume that S_1 and S_2 satisfy service instantiation condition, PS^{S_1} and PS^{S_2} are S_1 and S_2 's parameter declaration (input parameter or output parameter), or data type's value, or concept-description's value, or negotiation attribute's value respectively. Assume $\forall S \in PS^{S_i}, i = 1,2$, then the format of S is $< R : F >$, or $< R = F >$; as to PS^{S_1} and PS^{S_2}, a substitution σ is attribute-unifying substitution if and only if:

$$\forall R : (< R : F^{S_1} > \in PS^{S_1} \wedge < R : F^{S_2} > \in PS^{S_2}) \Rightarrow \sigma(F^{S_1}) = \sigma(F^{S_2});$$

or $\forall R : (< R = F^{S_1} > \in PS^{S_1} \wedge < R = F^{S_2} > \in PS^{S_2}) \Rightarrow \sigma(F^{S_1}) = \sigma(F^{S_2})$.

Definition 4(Service Instantiation) Given services description S_1, …, S_k and S_m, assume S_1, …, S_k and S_m satisfy service instantiation condition, σ_i is the attribute-unifying substitution of $S_i, 1 \le 1 \le k$ and S_m, then service S is the result of S_m instantiated by S_1, …, S_k if and only if :

(1) $A^S = A^{S_m}, SL^S = SL^{S_m}, CL^S = CL^{S_m}, SN^S = SN^{S_m}$;

(2) $SUP^S = \phi$;

(3) $Q^S = \sigma_1(Q^{S_1}) \bigcup \sigma_1(Q^{S_m}) \bigcup ... \bigcup \sigma_k(Q^{S_k}) \bigcup \sigma_k(Q^{S_m})$,where *Q* represent *CON*, *TY*, *CD*, *I*, *O*, S_I, S_O and S_{IO} respectively.

Because parent class of child class may have parent class too, so we may use the thinking of recursion to realize service instantiation. This instantiation algorithm is the following:

Service Instantiation Algorithm instantiate(S_m)

Input: service S_m

Output: service S instantiated by S_m

Algorithm: (1) $C = SUP^{S_m}, S \leftarrow S_m$;

(2) if $C = \phi$ return S;

(3) $\forall\ S_i \in C, C \leftarrow C - \{S_i\}$;

(4) $S_i \leftarrow instantiate(S_i)$;

(5) $A^S \leftarrow A^{S_m}, SL^S \leftarrow SL^{S_m}, CL^S \leftarrow CL^{S_m}, SN^S \leftarrow SN^{S_m}$;

(6) $SUP^S \leftarrow \phi, \sigma \leftarrow \phi$;

(7) $amend(CON^{S_m}, CON^{S_i}, \sigma), amend(TY^{S_m}, TY^{S_i}, \sigma), amend(I^{S_m}, I^{S_i}, \sigma),$
$amend(O^{S_m}, O^{S_i}, \sigma), amend(CD^{S_m}, CD^{S_i}, \sigma)$;

(8) $CON^S \leftarrow CON^{S_i}, TY^S \leftarrow TY^{S_i}, I^S \leftarrow I^{S_i}, O^S \leftarrow O^{S_i}, CD^S \leftarrow CD^{S_i}$;

(9) $S_I^S \leftarrow \sigma(S_I^{S_M}) \bigcup \sigma(S_I^{S_i}), S_O^S \leftarrow \sigma(S_O^{S_M}) \bigcup \sigma(S_O^{S_i}), S_{IO}^S \leftarrow \sigma(S_{IO}^{S_M}) \bigcup \sigma(S_{IO}^{S_i})$;

(10) *Goto* (2).

function $amend(PS^{S_1}, PS^{S_2}, \sigma)$

for $< R, F^{S_2} > \in PS^{S_2}$

if $\exists < R, F^{S_1} > \in PS^{S_1}$ *then*

$\sigma \leftarrow unify(F^{S_1}, F^{S_2}, \sigma), PS^{S_1} \leftarrow PS^{S_1} - < R, F^{S_1} >,$

$PS^{S_1} \leftarrow PS^{S_1} + < R, \sigma(F^{S_2}) >$

else

$PS^{S_1} \leftarrow PS^{S_1} + < R, F^{S_2} >$

Notation: $< R, F^{S_2} >$ represents $< R : F >$ or $< R = F >$, see Definition 3.

Definition 5(Type Subsumption) Given type t_1 and t_2, if t_2 subsumes t_1 (noted as $t_1 \sqsubseteq t_2$), then it is reasoned by the following inference rule[2]:

(1) if t_2 is type variable, then $t_1 \sqsubseteq t_2$;

(2) if $t_1 = t_2$ (i.e. t_1, t_2 represent same type), then $t_1 \sqsubseteq t_2$;

(3) if t_1 and t_2 are the type defined by terminology ontology, $t_1 \subseteq t_2$, then $t_1 \sqsubseteq t_2$, where $\subseteq$ interpreted in DL's semantics;

(4) if t_1 and t_2 are set, $t_1 \subseteq t_2$, then $t_1 \sqsubseteq t_2$, where $\subseteq$ interpreted in set theory;

(5) $t_1 \sqsubseteq t_1 \mid t_2$;

(6) $t_2 \sqsubseteq t_1 \mid t_2$;

(7) if $t_1 \sqsubseteq t_2$, $s_1 \sqsubseteq s_2$, then $(t_1, s_1) \sqsubseteq (t_2, s_2)$;

(8) if $t_1 \sqsubseteq t_2$, $s_1 \sqsubseteq s_2$, then $t_1 \mid s_1 \sqsubseteq t_2 \mid s_2$;

(9) if $t_1 \sqsubseteq t_2$, then $SetOf(t_1) \sqsubseteq SetOf(t_2)$;

(10) if $t_1 \sqsubseteq t_2$, then $ListOf(t_1) \sqsubseteq ListOf(t_2)$.

Definition 6 (Type Equality) Given type t_1 and t_2, if t_1 subsumes t_2 (i.e. $t_2 \sqsubseteq t_1$), and t_2 subsumes t_1 (i.e. $t_1 \sqsubseteq t_2$), then t_1 and t_2 equality, noted as $t_1 \equiv t_2$.

4.2.2.2 Approximate Service Matchmaking Algorithm

Approximate service matchmaking directly use the value of attribute **context** to finish it. Because there are two kinds of approximate service matchmaking: syntax based similar matchmaking and semantics based similar matchmaking, so there also are two kinds of approximate service matchmaking algorithms.

Algorithm 1 SYntax-Based-Similarity-Matchmaking SYBSM(SPS,SRS)

Input: Service description *SPS* of service provider *SP* and service description *SRS* of service requester *SR*

Output: Boolean value (true or false)

Algorithm: (1) if *SPS* contains attribute **isa**, then invoke function *instantiate*(*SPS)* to instantiate *SPS*. Assume the result instantiating *SPS* is service description *ISPS*, its value of attribute **context** is: $con^{SP} = \{C_1^{SP} * O_1^{SP},...,C_i^{SP} * O_i^{SP}\}$;

(2) if *SRS* contains attribute **isa**, then invoke function *instantiate*(*SRS)* to instantiate *SRS*. Assume the result instantiating *SRS* is service description *ISRS*, its value of attribute **context** is: $con^{SR} = \{C_1^{SR} * O_1^{SR},...,C_j^{SR} * O_j^{SR}\}$;

(3) if $|\{C_1^{SP},...,C_i^{SP}\} \bigcap \{C_1^{SP},...,C_j^{SR}\}| \Big/ |\{C_1^{SP},...,C_i^{SP}\} \bigcup \{C_1^{SP},...,C_j^{SR}\}| > \omega$, then return true; else return false, where threshold $\omega \in [0,1]$ given by users.

Example 1 Given two service descriptions S_1 and S_2, where S_1={service-name s1; context plane, ticket, book}, S_2={service-name s2; context fly, ticket, book}, because S_1.context={plane, ticket, book}, S_2.context={fly, ticket, book}, so S_1.context ∩ S_2.context={ticket, book}, S_1.context ∪ S_2.context ={plane, fly, ticket, book}, therefore $|S_1.context \cap S_2.context| / |S_1.context \cup S_2.context|$=2 / 4=0.5. Assume ω =0.7, then service descriptions S_1 and S_2 don't satisfy the syntax based approximate service matchmaking.

Algorithm 2 SEmantics-Based-Similarity-Matchmaking) SEBSM(SPS,SRS)

Input and output are the same as algorithm 1;

Algorithm: (1) and (2) are the same as algorithm 1;

(3) Given $\{O_1^{SP},...,O_i^{SP}\} \sqcap \{O_1^{SR},...,O_j^{SR}\} = \{O \mid (O = O_k^{SP} \wedge (\exists O_l^{SR}, O_k^{SP} \equiv O_l^{SR} \vee O_k^{SP} \subseteq O_l^{SR} \vee O_k^{SP} \supseteq O_l^{SR})) \vee (O = O_l^{SR} \wedge (\exists O_k^{SP}, O_l^{SR} \equiv O_k^{SP} \vee O_l^{SR} \subseteq O_k^{SP} \vee O_l^{SR} \supseteq O_k^{SP})), 1 \le k \le i, 1 \le l \le j\} = O_I$, if $|O_I| \Big/ |\{O_1^{SP},...,O_i^{SP}\} \bigcup \{O_1^{SR},...,O_j^{SR}\}| > \omega$, then return true; else return false, where threshold $\omega \in [0,1]$ given by users. $\equiv$, $\subseteq$ and $\supseteq$ interpreted in DL's semantics i.e., $O_l^{SR} \equiv O_k^{SP}$ represents O_l^{SR} and O_k^{SP} equality(i.e. $O_l^{SR^I} \equiv O_k^{SP^I}$); $O_l^{SR} \subseteq O_k^{SP}$ represents

O_k^{SP} subsume O_l^{SR} (i.e. $O_l^{SR^I} \subseteq O_k^{SP^I}$); $O_l^{SR} \supseteq O_k^{SP}$ represents O_l^{SR} subsumes O_k^{SP} (i.e. $O_l^{SR^I} \supseteq O_k^{SP^I}$).

Example 2 Given two service descriptions S_1 and S_2, where S_1={service-name s1; context plane*Plane, ticket*Ticket, book*Book}, S_2={service-name s2; context fly*Fly, ticket*Ticket, book*Book }, where Plane, Ticket, Book and Fly are terminology definition in description logic, Plane ≡ machine⊓can.fly, Ticket ≡ paper⊓function.seat, Book ≡ advanced-buy, Fly ≡ machine⊓can.fly. Because S_1.context.Ontology={Plane, Ticket, Book}, S_2.context.Ontology= {Fly, Ticket, Book}, and through description logic theory, it is known that Plane is equal to Fly, so S_1.context.Ontology ∩ S_2.context.Ontology ={Plane, Ticket, Book}, S_1.context.Ontology ∪ S_2.context.Ontology={Plane, Ticket, Book}, therefore |S_1.context.Ontology ∩ S_2.context.Ontology|/|S_1.context.Ontology ∪ S_2.context.Ontology|=3/3=1. Assume ω=0.7, then service descriptions S_1 and S_2 satisfy the semantics based approximate service matchmaking.

Remark: (1) Deciding two concepts whether satisfy relation ≡, ⊆ or ⊇ need knowledge of description logic DL.

(2) Deciding the relation between two concepts need the terminology ontology of the middle agent.

(3) The main difference between algorithm 1 and algorithm 2 is: Algorithm 1 only matches services in syntax level, i.e. it utilizes the keywords in **context** component(see section 3.1) to match services, but algorithm 2 uses ontology in **context** component(see section 3.1) to match services, i.e. it matches services in semantic level.

4.2.2.3 Precise Service Matchmaking Algorithm

Precise service matchmaking algorithm demands two services is equality in semantics level, and it uses the attribute value of **inputs**, **outputs**, **input-constraints**, **output-constraints** and **io-constrains** to realize service matchmaking.

Algorithm 3 Precise Matchmaking PM(SPS,SRS)

Input and output are the same as algorithm 1;

Algorithm: (1) If *SPS* attribute **isa**, then invoke function *instantiate* (*SPS)* to instantiate *SPS*. Assume the result instantiating *SPS* is service description *ISPS*, its attribute value of **inputs** is $I^{SP} = \{ iv_1^{SP} : it_1^{SP},..., iv_{i_{SP}}^{SP} : it_{i_{SP}}^{SP}\}$,the attribute value of **outputs** is $O^{SP} = \{ ov_1^{SP} : ot_1^{SP},..., ov_{j_{SP}}^{SP} : ot_{j_{SP}}^{SP}\}$, the attribute value of **input-constraints** is $S_I^{SP} = \{S_{I1}^{SP},..., S_{Ik_{SP}}^{SP}\}$, the attribute value of **output-constraints** is $S_O^{SP} = \{S_{O1}^{SP},..., S_{Ol_{SP}}^{SP}\}$, and the attribute value of **io-constrains** is $S_{IO}^{SP} = \{LS_{IO1}^{SP} \rightarrow RS_{IO1}^{SP},..., LS_{IOm_{SP}}^{SP} \rightarrow RS_{IOm_{SP}}^{SP}\}$;

(2) If *SRS* contains attribute **isa**, then invoke function *instantiate*(*SRS)* to instantiate *SRS* Assume the result of instantiating *SRS* is service description *ISRS*, its attribute value of **inputs** is $I^{SR} = \{ iv_1^{SR} : it_1^{SR},..., iv_{i_{SR}}^{SR} : it_{i_{SR}}^{SR}\}$,the attribute value of **outputs** is $O^{SR} = \{ ov_1^{SR} : ot_1^{SR},..., ov_{j_{SR}}^{SR} : ot_{j_{SR}}^{SR}\}$, the attribute value of **input-constraints** is $S_I^{SR} = \{S_{I1}^{SR},..., S_{Ik_{SR}}^{SR}\}$, the attribute value of **output-constraints** is $S_O^{SR} = \{S_{O1}^{SR},..., S_{Ol_{SR}}^{SR}\}$, the attribute value of **io-constrains** is $S_{IO}^{SR} = \{LS_{IO1}^{SR} \rightarrow RS_{IO1}^{SR},..., LS_{IOm_{SR}}^{SR} \rightarrow RS_{IOm_{SR}}^{SR}\}$;

(3) If $i_{SP} = i_{SR}$, and $it_g^{SP} \equiv it_g^{SR}, 1 \le g \le i_{SP}$, then goto(4), else return false;

(4) If $j_{SP} = j_{SR}$, and $ot_g^{SP} \equiv ot_g^{SR}, 1 \le g \le j_{SP}$, then goto(5), else return false;

(5) If exists a substitution σ, and $S_I^{SP} \Leftrightarrow \sigma(S_I^{SR})$, then goto(6), else return false;

(6) If exists a substitution σ, and $S_O^{SP} \Leftrightarrow \sigma(S_O^{SR})$, then goto(7), else return false;

(7) If exists a substitution σ, and $S_{IO}^{SP} \Leftrightarrow \sigma(S_{IO}^{SR})$, then return true; else return false;

(8) End.

Example 3 Given two service descriptions S_1 and S_2, where S_1={service-name s1; inputs xs:ListOf Real; outputs ys: ListOf Real; output-constraints before(x,y,ys)←ge(x,y)}, S_2= {service-name s2; inputs xs:ListOf Real; outputs ys: ListOf Real; output-constraints before(x,y,ys)←ge(x,y)}, then service descriptions S_1 and S_2 satisfy the precise service matchmaking. (for this example, please partially see [2]).

Remark: (1) Compare with algorithm 1 and algorithm 2, this algorithm utilizes the attribute value of **inputs**, **outputs**, **input-constraints**, **output-constraints** and **io-constrains** to match services.

(2) This algorithm need terminology ontology, see step (3) and step (4), and the relation between ontologies in **inputs** components or **outputs** components is equality relation; in addition, (5), (6) and (7) need not terminology ontology.

(3) Step (5) decide whether the input constrains equality, step (6) decide whether the output constrains equality, step (7) decide whether the input-output constrains equality. According to (2) and (3), this algorithm requires aforementioned equality relation, so it realizes the precise matchmaking.

4.2.2.4 Plug-in Service Matchmaking Algorithm

Plug-in service algorithm lines in middle between approximate match and exact match, and it demands service M provided by provider subsume service N provided by requester i.e., as to N, M can provide more service, and N can be "plugged in" M. It uses the attribute value of **inputs**, **outputs**, **input-constraints**, **output-constraints** and **io-constrains** to realize service matchmaking.

Algorithm 4 Plug-In Matchmaking PIM (SPS, SRS)

Input and output are the same as algorithm 1;

Algorithm: (1) and (2) are the same as algorithm 3;

(3) If $i_{SP} = i_{SR}$, and $it_g^{SP} \sqsupseteq it_g^{SR}, 1 \le g \le i_{SP}$, then goto(4), else return false;

(4) If $j_{SP} = j_{SR}$, and $ot_g^{SP} \sqsupseteq ot_g^{SR}, 1 \le g \le j_{SP}$, then goto(5), else return false;

(5) If exists a substitution σ, and $S_I^{SP} \Leftarrow \sigma(S_I^{SR})$, then goto(6), else return false;

(6) If exists a substitution σ, and $S_O^{SP} \Rightarrow \sigma(S_O^{SR})$, then goto(7), else return false;

(7) If exists a substitution σ, satisfy $\forall S_{IOi}^{SP} = LS_{IOi}^{SP} \rightarrow RS_{IOi}^{SP}$, $1 \le i \le m_{SP}$, $\exists S_{IOj}^{SR} = LS_{IOj}^{SR} \rightarrow RS_{IOj}^{SR}, 1 \le j \le l_{SR}$, and $(LS_{IOi}^{SP} \Leftarrow \sigma(LS_{IOj}^{SR})) \wedge (RS_{IOi}^{SP} \Rightarrow \sigma(RS_{IOj}^{SR}))$, then return true, else return false;

(8) End.

Example 4 Given two service descriptions S_1 and S_2, where S_1={service-name s1; inputs xs:ListOf Integer; outputs ys: ListOf Integer; input-constraints le(length(xs),100); output-constraints before(x,y,ys)←ge(x,y), in(x,ys)←in(x,xs)}, S_2={service-name s2; inputs xs:ListOf Real; outputs ys: ListOf Real; output-constraints before(x,y,ys)←ge(x,y), in(x,ys)←in(x,xs)}, then service descriptions S_1 and S_2 satisfy the plug-in service matchmaking, but they don't satisfy the precise service matchmaking. (for this example, please partially see [2]).

Remark: (1) This algorithm is same as algorithm 3, it also utilizes the attribute value of **inputs**, **outputs**, **input-constraints**, **output-constraints** and **io-constrains** to match services.

(2) This algorithm needs terminology ontology, see step (3) and step (4); the difference between algorithm 3 and this algorithm is: here is $\sqsubseteq$(subsumption relation), not $\equiv$(equality relation).

(3) Step (5) decide whether the input constrains satisfy imply relation, step (6) decide whether the output constrains satisfy imply relation, step (7) decide whether the input-output constrains satisfy imply relation. According to (2) and (3), this algorithm realizes the "plugged in" matchmaking.

In addition, all these service matchmaking algorithms are dynamic process, because Intelligent Web is open and dynamic, the service provider in Intelligent Web may register service, modify service and delete service anytime; at the same time, service requester may request its service and exit the Web anytime, therefore middle agent should dynamic update their records and match service dynamic.

5 Development of Matchmaking

5.1 Multi-agent Environment MAGE

MAGE (Multi-AGent Environment)[9] is a multi-agent environment with a collection of tools supporting the entire process of agent-oriented software engineering and programming. It is designed to facilitate the rapid design and development of new multi-agent applications by abstracting into a toolkit the common principles and components underlying many multi-agent systems. The idea was to create a relatively general purpose and customizable toolkit that could be used by software users with only basic competence in agent technology to analyze, design, implement and deploy multi-agent systems. MAGE toolkit has the following features: (1) The toolkit utilize standardized technology and standardized specification wherever feasible, for examples our MAGE is FIPA compliant, and agent interaction is based on FIPA-ACL. (2) The toolkit provide agent developers with all the domain-independent functionality such that they need only implement the domain-specific problem solving abilities of the agents they want to define. (3) The toolkit provide friendly and easy-to-use human-computer interface by the visual programming paradigm and the pick-and-choose principle. (4) The toolkit support an open and flexible design so that it is easily to extend, thus users can easily add to the library different kind of components.

5.2 Implement

In multi-agent environment MAGE, we have realized the service description language SDLSIN and the four different service matchmaking algorithms in Java. We realized a DF (Directory Facilitator)

agent and SBroker (Service Broker) agent in MAGE, where DF agent stores agent's capability (service) description and finishes service matchmaking, and SBroker agent finishes service invoke, service negotiation, service composition, service cooperation and service control, etc. DF and SBroker make up of the middle agent of our MAGE. Service providers register their service to DF agent in SDLSIN. Service requester puts in its request to SBroker in SDLSIN, then SBroker act as this requester to finish all service reasoning work (such as service matchmaking, service negotiation, service composition), at last SBroker returns the answer to this requester. Because service reasoning needs DL reasoning and FOPL reasoning, so we realized a DL reasoner (DLRM) and a FOPL reasoner (FOPLReasoner) in Java at first.

In open, heterogeneous, and dynamic Intelligent Web, how to find the agent that it need as to requester? It is a good idea that import middle agent. In MAGE services environment, we developed DF and SBroker agents as middle agent. MAGE services environment architecture sees Figure 1, and Figure 2 is the interface of our MAGE services environment.

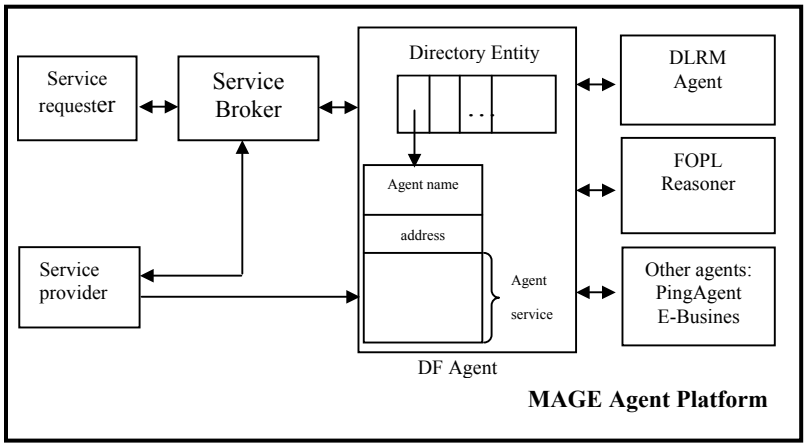

Figure 1: MAGE services environment architecture

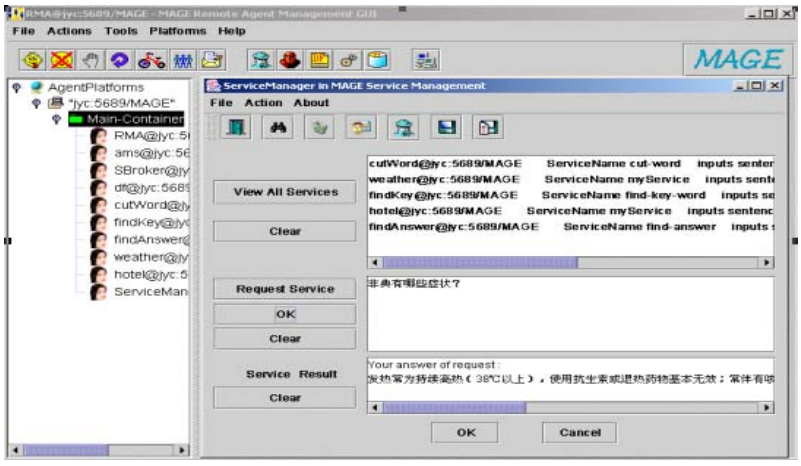

Figure 2: MAGE services environment interface

In MAGE services environment, because DF agent has four different service matchmaking algorithms, so service requester may select one based on its practical need (as to which algorithm is adopted, it is decided by service requester). For example, if service requester only needs some approximate services, it can select approximate service matchmaking algorithm. At present, MAGE

services environment is developed in Java, and it need not any database. DF agent stores service description in memory. If there are many services in DF agent in practical application, we will use database to store service description. In this MAGE services environment, we implement two application systems, one is knowledge about SARS service system, and the other is integration information about Beijing service system. In the first system, there are three kinds of agents, i.e. cut word agent, find pattern agent, and find answer agent. There are seven kinds of agents, i.e. predict weather agent, book ticket agent, find traffic agent, find shop agent, find food agent, find hotel agent and find entertainment agent in the second system. Any service requester may put in service request in natural language to SBroker agent, then SBroker agent will change this request into SDLSIN format, and SBroker agent will search appropriate service provider and acquire the answer to service request. At last SBroker agent send the answer to service requester.

In the aforementioned two application systems, the performance is satisfactory, and the algorithms produce the expected results. Because there isn't similar system now, we can't compare our system with others systems on performance and result. As to the different values of ω, when the value of ω is smaller, more service providers can be found. At present, if there are many agents, which may provide same service in MAGE services environment, the first agent will be selected. We will improve this situation through service negotiation.

Discussion: In current implement, we design a description logic agent DLRM and a first order predicate logic agent FOPLReasoner, and these two agents assist DF agent to finish service matchmaking. With the development with MAGE, we may build description logic reasoning and first order logic reasoning in DF agent. On the other hand, we may add several DF agents in MAGE in order to improve the efficiency of service matchmaking.

6 Related Work

Agent service description and service matchmaking have been actively studied since the inception of intelligent agent research. The earliest matchmaker was the ABSI facilitor[10], which was based on the KQML specification and used the KIF as the content language. The matchmaking between the advertisement and request expressed in KIF was the simple unification with equality predicate.

Kuokka and Harada[11] presented the SHADE and COINS systems for matchmaking. The matchmaking algorithm of COINS used the TF-IDF and used free text as the content language. The matchmaking algorithm used in SHADE was a Prolog-like unification. The content language of SHADE consisted of two parts, one was a subset of FIF; the other was a structured logic representation called MAX. They didn't present the concept of service description language and didn't consider the matchmaking in semantics level.

R. J. Bayardo, et al[12] presented a service broker-based information system InfoSleuth. Its content language supported by InfoSleuth was KIF and the deductive database language LDL++. The matchmaking algorithm used was constraint matching, which was an intersection function between the user query and the resource constraints. They also didn't present the service description language and its matchmaking algorithm was limited.

G J Wickler[3] studied and developed the agent capability description language CDL which was a action language, but it was a syntax-based language and didn't consider the negotiation mechanism of service; on the other hand, CDL didn't support the definition of data types. Logic-based reasoning over description in CDL was based on the notion of subsumption. It couldn't present the approximate matchmaking.

K Arisha, et al[4] also presented a kind of agent service description language SDL which was a single HTML-like language. This language used hierarchy mechanism and synonym dictionary to study the semantics of service description. The matchmaking algorithms were k-nearest neighbor request and δ-nearest search using semantics instance, so it only can present approximate algorithm.

K Sycara, et al[2] studied and developed the agent capability description language LARKS. This language considered the trade-off between Quality of Service (QoS) and efficiency of service, but it didn't consider the inheritance mechanism of service description and agent state language. More important it didn't consider the negotiation mechanism of service.

W3C also presented a kind of Web Services Description Language WSDL [13]. It is an XML format for describing network service as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. But it is a syntax based description language.

Based on these reasons, we presented a kind of agent service description language SDLSIN with semantics and inheritance and supporting negotiation. This language not only considered semantic service description of agent, but also considered the inheritance and negotiation mechanism of agent service description, agent state language, and data types, therefore it overcome the insufficiencies of LARKS, CDL and SDL. Based on SDLSIN, we studied different dynamic service matchmaking algorithms and strategies using SDLSIN in Intelligent Web.

7 Conclusions and Future Work

In this paper we studied the agent service description and matchmaking problems, and analyzed the insufficiencies of the service description language CDL, SDL and LARKS. Combining with the characteristics of Intelligent Web, Web Services and Grid Services, we presented a kind of agent service description language SDLSIN with semantics and inheritance and supporting negotiation. This language not only considered semantic service description of agent, but also considered the inheritance and negotiation mechanism of agent service description, agent state language, and data types. Based on SDLSIN, we studied different dynamic service matchmaking algorithms and strategies using SDLSIN in Intelligent Web.

Future work includes: study how to add many middle agents to improve the service management work and coordinate them; study how to develop a kind of lightweight service description language adapting to pervasive computing; and study the theoretic model and its formal semantics of service management in Intelligent Web.

Acknowledgements

This research is supported by High-Tech Program 863 (2001AA113121, 2003AA115220) and National Natural Science Foundation of China (90104021).

The first version (some result) of this paper has been published by Acta Electronica Sinica(Jiang Yuncheng, et al. Dynamic Service Matchmaking in Multi-Agent Systems, 2004(3): 457-461, in Chinese).

References

1. Zhongzhi Shi, Mingkai Dong, Yuncheng Jiang et al. Intelligent Web, *Computer Science*, 30(9), pp. 1-4, 2003. (in Chineses).

2. Katia Sycara, Seth Widoff, Matthias Klusch and Jianguo Lu. LARKS: Dynamic Matchmaking Among Heterogenous Software Agents in Cyberspace. *Autonomous Agents and Multi-Agent Systems*, 5(2), pp. 173-203,2002.

3. Gerhard Jurgen Wickler . Using Expressive and Flexible Action Representations to Reason about Capabilities for Intelligent Agent Cooperation. Ph.D. thesis, University of Edinburgh, 1999.

4. Khaled Arisha, Sarit Kraus, Fatma Ozcan, Robert Ross, and V.S. Subrahmanian. IMPACT: The Interactive Maryland Platform for Agents Collaborating Together. *IEEE Intelligent Systems,* MarApr issue 14(2), pp. 64-72, 1999.

5. Javier Gonzalez-Castillo, David Trastour, Claudio Bartolini,Description Logics for Matchmaking of Services,*Workshop on Applications of Description Logics*, Vienna, Austria ,September 18, 2001.

6. F. Baader, W. Nutt. Basic description logic. In F. Baader et al, editor, *The Description Logic Handbook*: *Theory, Implementation and Applications.* Cambridge University Press, 2002.

7. M. Klusch and K. Sycara, Brokering and Matchmaking for Coordination of Agent Societies: A Survey, in: A. Omicini, F. Zambonelli, M. Klusch, and R. Tolksdorf (Eds.), *Coordination of Internet Agents: Models, Technologies and Applications*, Springer, 2001, Chapter 8.

8. H. C. Wong and K. Sycara, A Taxonomy of Middle Agents for the Internet, *Proceedings of 4th ICMAS*, IEEE Computer Society Press, 2000, pp. 465-466.

9. http://www.intsci.ac.cn/en/research/mage.html.

10. N. Singh, A Common Lisp API and Facilitator for ABSI: Version 2.0.3, *Technical Report Logic-93-4*, Logic Group, Computer Science Department, Stanford University, 1993.

11. 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. 239-245.

12. R. J. Bayardo, W. Bohrer, R. Brice et al., InfoSleuth: Agent-Based Semantic Integration of Information in Open and Dynamic Environments, in: M. N. Huhns and M. P. Singh (Eds.), *Readings in Agents*, Morgan Kaufmann, CA, 1998, pp. 205-216.

13. http://www.w3.org/TR/wsdl12/. Web Services Description Language (WSDL) Version 1.2. W3C Working Draft 9 July 2002.