Fundamenta Informaticae 96 (2009) 211–226
DOI 10.3233/FI-2009-175
IOS Press

A Semantic Model for Matchmaking of Web Services Based on Description Logics

Guohua Shen[*][†]
College of Information Science and Technology
Nanjing University of Aeronautics and Astronautics
29 Yudao St. Nanjing, China
ghshen@nuaa.edu.cn, ghshen.cn@gmail.com

Zhiqiu Huang, Yuping Zhang, Xiaodong Zhu, Jun Yang
College of Information Science and Technology
Nanjing University of Aeronautics and Astronautics
Nanjing, China

Abstract. Matchmaking plays an important role in Web services interactions. The matchmaking based on keywords easily leads to low precision, Meanwhile, the current semantic service discovery methods perform service I/O based profile matching, there exists no matchmaker that performs an integrated service matching by additional reasoning on logically defined preconditions, effects, Qos and so on. In this paper, the semantic web services are described based on Description Logics, and the services description model is designed, which describes the various aspects (such as IOPEs, Qos and so on) of the web services. So the services matchmaking is transformed into the match of concepts. The service match algorithm is proposed and the description logics reasoner RacerPro is adopted for Web services discovery. We show how the semantic matching between providers and a requester is performed by a case study.

Keywords: Web Services, matchmaking, description logics, Semantic web, ontology

[*]Address for correspondence: College of Information Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, 210016, China
[†]We thank the anonymous referee for helpful suggestions that lead us to improve the paper. And thank the editors in *Fundamenta Informaticae* for their kindness and responsibility.

1. Introduction

Web services are software-powered resources or self-contained functional components whose capabilities can be accessed over the Internet. Web services provide a standard means of interoperating between different software applications, running on a variety of platforms and/or frameworks. How services should be modeled, which facilitates finding services, is the key issues.

The semantics of a Web service is the contract between the service requester and the provider regarding the purpose and consequences of the interaction. While the service description represents a contract governing the mechanics of interacting with a particular service, the semantics represents a contract governing the meaning and purpose of that interaction.

Discovery is the act of locating a machine-processable description of a Web service that may have been previously unknown and that meets certain functional criteria. The goal is to find an appropriate Web service [1].

The keyword-based methods, like UDDI, are simple but easily lead to low precision because of lacking semantic. Semantic web services are the combination of web services and semantic web, which makes the service description machine-understandable and -processable. The main semantic services models are OWL-S [2] (formerly DAML-S) and WSMO [3]. Sycara implemented the DAML-S/UDDI matchmaker that expands on UDDI by providing semantic capability matching [4]. The methods mentioned in [5, 6, 7] are based on OWL-S, while the method in [8] is based on WSMO.

Meanwhile, the current semantic service discovery methods perform service I/O based profile matching, there exists no matchmaker that performs an integrated service matching by additional reasoning on logically defined preconditions, effects, Qos, basic information and so on [7].

Description logics (DL), which are decidable fragments of first order logic (FOL), form a family of languages for modeling an application domain in terms of objects, classes and relationships between classes, and for reasoning about them. DL offer considerable expressive power, while reasoning is still decidable [9].

We propose a semantic service description model, which works out all aspects of service capabilities (IOPEs, Qos and so on) at an abstract level, based on the use of description logics. The strength of our work is that it provides rigorous way to model services, and the DL reasoner can be used to reason about service matching. We describe the semantic of web services as the several aspects, which are expressed as DL concepts in TBox. The services matching can be made by concept subsumption reasoning.

The paper is organized as follows: in next section, description logics and DL based semantic web services model are represented. In section 3 a service matchmaking approach and its algorithm based on DL reasoning are proposed. A case study, which selects RacerPro as a DL reasoner, is demonstrated in section 4. The related work and conclusion follow in Section 5 and 6.

2. Description logics and semantic web service models

DL play an important role in the Semantic Web since they are the basis of the OWL, which is recommended by W3C. A knowledge base K of DL is constituted by the TBox T and the ABox A, denoted as $K = (T, A)$, where the TBox introduces the *terminology*, i.e., the vocabulary of an application domain, while the ABox contains *assertions* about named individuals in terms of this vocabulary. The basic elements of Description Logics are concepts and roles. Arbitrary concepts C is built according to the following syntax rules [9]:

C::=$A|\bot|\top|\neg A|C_1 \sqcap C_2|C_1 \sqcup C_2|\forall R.C|\exists R.C|\theta nR.C|\{a_1,...,a_n\}$

Where $\bot$ is bottom concept, $\top$ is top concept, θ={$\leq$, $\geq$, =}, and $\{a_1,...,a_n\}$ is a set (or one-of) constructor. In addition, DL provide the description and reasoning of concrete domain, such as integer, real and string etc. A concrete domain $\mathcal{D}$ consists of a set $\Delta^{\mathcal{D}}$, the domain of $\mathcal{D}$, and a set pred($\mathcal{D}$), the predicate names of $\mathcal{D}$. $\mathcal{D}$ is called admissible iff (1) the set of its predicate names is closed under negation and contains a name $\top_{\mathcal{D}}$ for $\Delta^{\mathcal{D}}$, and (2) the satisfiability problem for $\mathcal{D}$ is decidable.

The reasoning in DL consists of TBox reasoning and ABox reasoning. The reasoning tasks for concepts consist of satisfiability, subsumption, equivalence, disjointness and so on [9].

2.1. Semantic web service model

Following OWL-S model and WSMO, the semantic model of web service for matchmaking is defined as follows. The model is highly abstract and ignores some details, such as binding, invoke, transaction and exception handling.

Definition 2.1. (semantic web service meta-model)
The semantic web service meta-model is a 3-tuple, *SWSDescription* = (*Ontology, Profile, Capability*), where *Ontology* describes the set of basic terms for service semantic, *Profile* presents the general information of the service, and *Capability* depicts what the service can do. The service model can be expressed in DL as follows:

SWSDescription ::= ($\exists$*hasProfile*.*Profile*)$\sqcap$($\exists$*hasCapability*.*Capability*)

Both *Profile* and *Capability* are described as concepts in DL, and see definition 2.2 and 2.3 for their details.

Definition 2.2. (service profile)
The service profile is a 2-tuple, *Profile* = (*Basic*, *Qos*), where *Basic* expresses the basic information like service name, category, creator, version etc, and *Qos* describes the non-functional properties like cost, response time.

The service profile can be expressed in DL as follows:

Profile ::= ($\exists$*hasBasic*.*Basic*)$\sqcap$ ($\exists$*hasQos*.*Qos*)

Basic ::= ($\exists$*serviceName*.$\top_{\mathcal{D}}$)$\sqcap$ ($\exists$*serviceCategory*.*Category*)$\sqcap$($\exists$*author*.*Person*) $\sqcap$($\exists$*version*.$\top_{\mathcal{D}}$)

Qos ::= ($\exists$*cost*. $\top_{\mathcal{D}}$) $\sqcap$ ($\exists$ *responseTime*. $\top_{\mathcal{D}}$)

Where $\mathcal{D}$ is a concrete domain and $\top_{\mathcal{D}}$ is a name for the domain of $\mathcal{D}$. For example, The concrete domain $\mathcal{N}$ has the set **N** of all nonnegative integers as its domain, and pred($\mathcal{N}$) consists of the binary predicate names $\leq$, $\geq$ as well as the unary predicate names $\leq_n$, $\geq_n$ for n$\in$**N**.

Definition 2.3. (service capability)
The service capability is a 4-tuple, *Capability* = (*Input, Output, Precondition, Effect*), where *Input, Output, Precondition, Effect* of the service (called IOPEs) describe the inputs required by the service, the outputs generated, external conditions require to be satisfied, and the effect of changing such conditions respectively.

The service capability can be expressed in DL as follows:

Capability::=($\exists$*hasPrecondition*.*Precondition*)$\sqcap$($\exists$ *hasInput*.*Input*)$\sqcap$($\exists$*hasOutput*.*Output*)$\sqcap$ ($\exists$*hasEffect*. *Effect*)

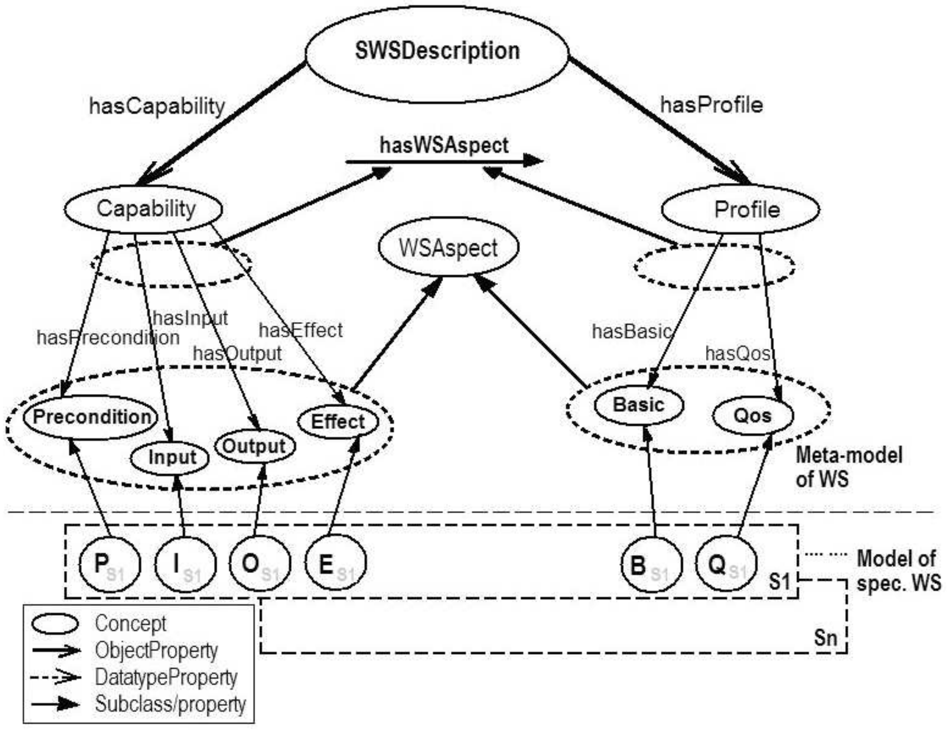

Figure 1. Semantic web services description model.

The semantic meta-model of web services is shown in Figure 1, where *Basic, Qos, Precondition, Input, Output, Effect* are modeled as concepts, which corresponds to the aspect defined in Definition 2.2 and 2.3. And all these are the sub-concept of *WSAspect*, which are expressed in DL as follows:

$Basic \sqsubseteq WSAspect$, $Qos \sqsubseteq WSAspect$,

$Input \sqsubseteq WSAspect$, $Output \sqsubseteq WSAspect$, $Effect \sqsubseteq WSAspect$, $Precondition \sqsubseteq WSAspect$

The roles *hasBasic*, *hasQos*, *hasPreconditon*, *hasInput*, *hasOutput* and *hasEffect* are the sub-role of *hasWSAspect*, expressed as :

$hasBasic \sqsubseteq hasWSAspect$, $hasQos \sqsubseteq hasWSAspect$,

$hasInput \sqsubseteq hasWSAspect$, $hasOutput \sqsubseteq hasWSAspect$, $hasPreconditon \sqsubseteq hasWSAspect$

Example 2.1. (EuroTravel)
A *EuroTravel* (*ET* for short) Service requires as a precondition a valid visa card and origin and destination in *austria* and *italy*, and as input the visa card and price. As output it generates a confirmation, and as effect the card is charged.

Based on the semantic service meta-model, the aspects of the specific service can be expressed by inheritance. And these generate the model of the service (or called service class), which is shown in the bottom of Figure 1. For example, the service *ET* (see example 2.1) requires input as the price, visa card, origin and destination. So its input can be derived from concept *Input*.

Input$_{ET}$::=*Input*$\sqcap$($\exists$ *origin.EuroCountry*)$\sqcap$($\exists$ *destination.EuroCountry*)$\sqcap$($\exists$ *price*.$\top_{\mathcal{D}}$) $\sqcap$ ($\exists$*hasCard. VisaCard*)

The *Output, Preconditions, Effects* etc. of the service *ET* can be described in the same way.

The capabilities of specific service depend on the domain knowledge, which is expressed in knowledge base using ontologies and axioms. For example, the service *ET* is described in terms of the knowledge like: *EuroCountry* is a sub-concept of the *Country*, and *austria* and *italy* are instances of the *EuroCountry*, and expressed in DL as follows:

EuroCountry $\sqsubseteq$ *Country*

EuroCountry (*austria*), *EuroCountry* (*italy*)

We distinguish between a concrete service instance and an abstract service class. A service instance defines all details of a business interaction, and is individual of the service class. An service class acts as a template for service instances. Our approach is focused on the meta-model and model of the web service instead of the instance, and the meta-model and model are expressed mainly in TBox of the knowledge base.

The intuitive semantic of the web service S capabilities is that: given the input I under the precondition P, the output O will be generated and with the effects E. Let $I(x_1,...,x_n)$, $O(x_1,...,x_n)$, $P(x_1,...,x_n)$, $E(x_1,...,x_n)$ be the IOPEs formulas in service capabilities, with free variables in $\{x_1,...,x_n\}$. We could write down the relation between these formulas as follows[3].

$\forall x_1,...,x_n \quad P(x_1,...,x_n) \wedge I(x_1,...,x_n) \rightarrow_S O(x_1,...,x_n) \wedge E(x_1,...,x_n)$ (4.1)

Each formula in form (4.1) uses the same free variables within different aspect of the capability means referring to the same entity. If the basic information B and quality Q of the service S are also taken into consideration, the form (4.1) will be extended like form (4.2).

$\forall x_1,...,x_n \quad B(x_1,...,x_n) \wedge Q(x_1,...,x_n) \wedge P(x_1,...,x_n) \wedge I(x_1,...,x_n) \rightarrow_S O(x_1,...,x_n) \wedge E(x_1,...,x_n)$ (4.2)

Our approach describes the different aspects of the service (*Basic, Qos, IOPEs*) and they are in the level of the service class. For example, the *IOPEs* of the service *ET* can be expressed as:

P_{ET}::=*Precondition*$\sqcap$($\forall$*origin*.{*austria,italy*})$\sqcap$($\forall$*destination*.{*austria,italy*})$\sqcap$($\exists$*hasCard. VisaCard*)

I_{ET}::=*Input*$\sqcap$($\exists$ *origin.EuroCountry*) $\sqcap$ ($\exists$ *destination. EuroCountry*) $\sqcap$ ($\exists$ *price*.$\top_{\mathcal{D}}$)$\sqcap$ ($\exists$*hasCard.VisaCard*)

O_{ET}::=*Output*$\sqcap$ ($\exists$ *hasConfirmation.Confirmation*)

E_{ET}::=*Effect*$\sqcap$ ($\exists$ *hasCharge.Charge*)

In the definition of P_{ET}, $\forall$*origin*.{*austria, italy*} means all the origins are restricted in *austria* or *italy*, which is written using set (or one-of) constructor.

2.2. Semantic web service matching

Finding the semantic web service is the process of service matchmaking.

Definition 2.4. (service matchmaking)
Service matchmaking is a process that requires a repository host to take a service requester as input and to return all service providers which may potentially satisfy the requirements specified in the input requester, written as:

matches(R)=$\{P \in \alpha |$ *compatible*(P, R) $\}$ (4.3)

Let α be the set of all providers in a given repository. For a given requester R, the matchmaking algorithm of the repository host returns the set of all providers which are compatible. *compatible*(P, R) means that service P is compatible with R (see definition 2.5). Note that the service R and P here are the service description class based on our semantic service model.

Definition 2.5. (service compatibility)
The service P is compatible with service R, written *compatible*(P, R) or $P \succeq R$, if B_R $(x_1,...,x_n) \wedge Q_R$ $(x_1,...,x_n) \wedge P_R$ $(x_1,...,x_n) \wedge I_R$ $(x_1,...,x_n) \rightarrow_R O_R$ $(x_1,...,x_n) \wedge E_R$ $(x_1,...,x_n)$, then B_R $(x_1,...\ ,x_n) \wedge$ Q_R $(x_1,...,x_n) \wedge P_R$ $(x_1,...,\mathrm{x}_n) \wedge I_R$ $(x_1,...,x_n) \rightarrow_P O_R$ $(x_1,...,x_n) \wedge E_R$ $(x_1,...,x_n)$.

That is, if P has the same pre-aspects as R, then P must generate the same post-aspects as R.
Note that $P \succeq R$ means P could be plugged in place of R.
Our model of the web service describes all the aspects of service, from which we derive the compatibility conditions.

Theorem 2.1. (compatibility conditions)
Let P be service provider, and R be service requester. If $\forall x_1,...,x_n$ $(B_P(x_1,...,x_n) \leftarrow B_R(x_1,...,x_n))$, and $\forall x_1,...,x_n$ $(Q_P(x_1,...,x_n) \leftarrow Q_R(x_1,...,x_n))$, and $\forall x_1,...,x_n$ $(P_P(x_1,...,x_n) \leftarrow P_R(x_1,...,x_n))$, and $\forall x_1,...,x_n$ $(I_P(x_1,...,x_n) \leftarrow I_R(x_1,...,x_n))$, and $\forall x_1,...,x_n(O_P(x_1,...,x_n) \rightarrow O_R(x_1,...,x_n))$, and $\forall x_1,...,x_n$ $(E_P(x_1,...,x_n) \rightarrow E_R(x_1,...,x_n))$ are satisfied, then $P \succeq R$.

The theorem 2.1 can be expressed as follows:
$(\forall x_1,...,x_n$ $(B_P(x_1,...,x_n) \leftarrow B_R(x_1,...,x_n))) \wedge (\forall x_1,...,x_n$ $(Q_P(x_1,...,x_n) \leftarrow Q_R(x_1,...,x_n))) \wedge$ $(\forall x_1,...,x_n$ $(P_P(x_1,...,x_n) \leftarrow P_R(x_1,...,x_n))) \wedge (\forall x_1,...,x_n$ $(I_P(x_1,...,x_n) \leftarrow I_R(x_1,...,x_n))) \wedge (\forall x_1,...,x_n$ $(O_P(x_1,...,x_n) \rightarrow O_R(x_1,...,x_n))) \wedge (\forall x_1,...,x_n(E_P(x_1,...,x_n) \rightarrow E_R(x_1,...,x_n))) \quad \Rightarrow \quad P \succeq R$
Where all the aspects of the service are written in concepts, and there is a difference, called "satisfy-direction difference" [10], between inputs and outputs matching.

Proof:
All the condition clauses in theorem 2.1 are as follows:
(1) $\forall x_1,...,x_n$ $(B_P(x_1,...,x_n) \leftarrow B_R(x_1,...,x_n))$
(2) $\forall x_1,...,x_n$ $(Q_P(x_1,...,x_n) \leftarrow Q_R(x_1,...,x_n))$
(3) $\forall x_1,...,x_n$ $(P_P(x_1,...,x_n) \leftarrow P_R(x_1,...,x_n))$
(4) $\forall x_1,...,x_n$ $(I_P(x_1,...,x_n) \leftarrow I_R(x_1,...,x_n))$
(5) $\forall x_1,...,x_n$ $(O_P(x_1,...,x_n) \rightarrow O_R(x_1,...,x_n))$
(6) $\forall x_1,...,x_n$ $(E_P(x_1,...,x_n) \rightarrow E_R(x_1,...,x_n))$
According to the definition of service compatibility (see definition 2.5), for service P $^{(*)}$, if P has the pre-aspects as B_R, Q_R, ...,I_R $^{(**)}$, and the compatibility conditions $^{(***)}$ are satisfied, then P has the post-aspects as O_R, E_R $^{(****)}$. The proof rule is as follows:

$$\frac{(B_P \wedge Q_P \wedge P_P \wedge I_P \rightarrow O_P \wedge E_P), B_R, Q_R, P_R, I_R, (B_P \leftarrow B_R), ..., (E_P \rightarrow E_R)}{O_R, E_R}$$

Where the formulas are written in the simple way, e.g., B_P stands for $B_P(x_1,...,x_n)$.
The formulas marked with $^{(*)}$, $^{(**)}$, $^{(***)}$, $^{(****)}$correspond to clauses (8), (7), (1)-(6), (9'). That is:
(7) $B_R(x_1,...,x_n)$ $\wedge Q_R(x_1,...,x_n)$ $\wedge P_R(x_1,...,x_n)$ $\wedge I_R(x_1,...,x_n)$

(8) $\forall\, x_1,...,x_n\ B_P(\mathrm{x}_1,...,x_n) \wedge Q_P(x_1,...,x_n) \wedge P_P(x_1,...,x_n) \wedge I_P(x_1,...,x_n) \rightarrow O_P(x_1,...,x_n) \wedge$ $E_P\ (x_1,...,x_n)$

(9') $O_R\ (x_1,...,x_n) \wedge E_R\ (x_1,...,x_n)$

The theorem can be proved by the resolution principle. The resolution principle, due to Robinson, is a method of theorem proving that proceeds by constructing proofs by contradiction. Generation of a resolvent from two clauses, called resolution, is the sole rule of inference of the resolution principle. The resolution principle is complete, so a set (conjunction) of clauses is unsatisfiable iff the empty clause can be derived from it by resolution.

According to the resolution principle, the theorem is true when the set of clauses S={(1),(2), . . . , (8), ¬(9')} is unsatisfiable. Let (9)= ¬(9'), eliminate the universal quantification(∀) and implication(→), we get the clauses in S as follows, where (7.1),. . . , (7.4) are from clause (7), (8.1), (8.2) are from clause (8), and each clause is in disjunctive normal form.

(1) $B_P(x_1,\ldots,x_n) \vee \neg B_R(x_1,\ldots,x_n)$
(2) $Q_P(x_1,\ldots,x_n) \vee \neg Q_R(x_1,\ldots,x_n)$
(3) $P_P(x_1,\ldots,x_n) \vee \neg P_R(x_1,\ldots,x_n)$
(4) $I_P(x_1,\ldots,x_n) \vee \neg I_R(x_1,\ldots,x_n)$
(5) $\neg O_P(x_1,\ldots,x_n) \vee O_R(x_1,\ldots,x_n)$
(6) $\neg E_P(x_1,\ldots,x_n) \vee E_R(x_1,\ldots,x_n)$
(7.1) $B_R(x_1,\ldots,x_n)$
(7.2) $Q_R(x_1,\ldots,x_n)$
(7.3) $P_R(x_1,\ldots,x_n)$
(7.4) $I_R(x_1,\ldots,x_n)$
(8.1) $\neg B_P\ (x_1,\ldots,x_n) \vee \neg Q_P\ (x_1,\ldots,x_n) \vee \neg P_P\ (x_1,\ldots,x_n) \vee \neg I_P\ (x_1,\ldots,x_n) \vee O_P\ (x_1,\ldots,x_n)$
(8.2) $\neg B_P\ (x_1,\ldots,\mathrm{x}_n) \vee \neg Q_P\ (x_1,\ldots,x_n) \vee \neg P_P\ (x_1,\ldots,x_n) \vee \neg I_P\ (x_1,\ldots,x_n) \vee E_P\ (x_1,\ldots,x_n)$
(9) $\neg O_R(x_1,\ldots,\mathrm{x}_n) \vee \neg E_R(x_1,\ldots,x_n)$

Applying resolution principle, we get:
(10) $B_P(x_1,\ldots,\mathrm{x}_n)$ by(1), (7.1)
(11) $Q_P(x_1,\ldots,\mathrm{x}_n)$ by (2), (7.2)
(12) $P_P(x_1,\ldots,\mathrm{x}_n)$ by (3), (7.3)
(13) $I_P(x_1,\ldots,\mathrm{x}_n)$ by (4), (7.4)
(14) $O_R\ (x_1,\ldots,\mathrm{x}_n)$ by (5), (8.1), (10), (11), (12), (13)
(15) $E_R\ (x_1,\ldots,\mathrm{x}_n)$ by (6), (8.2), (10), (11), (12), (13)
(16) □ by (9), (14), (15)

We can obtain empty clause with S, so S is unsatisfied. That is, clause (9') is the logic result of the clauses (1),(2), . . . , (8). The theorem is proved. □

Note that the service compatibility is not symmetrical, that is, $P \succeq R \neq R \succeq P$

The implication (→) in logic is expressed as subsumption ($\sqsubseteq$) in DL. So the theorem 2.1 can be described as:

Corollary 2.1. (compatibility conditions)
Service provider P is compatible with service requester R, if B_P subsumes B_R, and Q_P subsumes Q_R, and P_P subsumes P_R, and I_P subsumes I_R, and O_P is subsumed by O_R, and E_P is subsumed by E_R.

The Corollary 2.1 can be expressed as follows:
$(B_P \sqsupseteq B_R) \wedge (Q_P \sqsupseteq Q_R) \wedge (P_P \sqsupseteq P_R) \wedge (I_P \sqsupseteq I_R) \wedge (O_P \sqsubseteq O_R) \wedge (E_P \sqsubseteq E_R) \Rightarrow P \succeq R$

Definition 2.6. (degree of match)
The rational for the degree assignment[5] is described below:

(1) Exact: if requester R is compatible with provider P, and P is compatible with R. We call the match *exact*, written $R \equiv P$.

(2) PlugIn: if provider P is compatible with requester R, we call the match *Plugin*, written $P \succeq R$.

(3) Subsume: if requester R is compatible with provider P, we call the match *subsume*, written $P \preceq R$.

(4) Fail: otherwise we call *fail*.

Obviously, the requester expects first and foremost that the provider achieves the output requested at the highest degree. The degrees of *exact* and *plugin* are satisfied that P is compatible with R. The degree of *subsume* is not satisfied (in fact, R is compatible with P), but it shows that P is close to R.

3. Reasoning about service matchmaking

3.1. Modeling web service in DL

DL is fit for the service match, because that basic elements (concepts and roles) provided by DL are suitable for modeling static object, and that the DL system offers services that reason about them.

We can express the semantic web service meta-model in DL according to definition 2.1, 2.2 and 2.3. First, the top level ontologies are defined as follows:

SWSDescription ::= (∃hasProfile.*Profile*)⊓(∃hasCapability.*Capability*),

Profile ::= (∃hasBasic.*Basic*)⊓ (∃hasQos.*Qos*), *Basic*⊑*WSAspect*, *Qos*⊑*WSAspect*,

Capability::=(∃hasPrecondition.*Precondition*)⊓(∃hasInput.*Input*)⊓(∃hasOutput.*Output*)⊓(∃hasEffect. *Effect*),

Input⊑*WSAspect*, *Output*⊑*WSAspect*, *Effect*⊑*WSAspect*, *Precondition*⊑*WSAspect*,...

Next, the common knowledge and domain knowledge of the service are described in DL knowledge base by the domain experts, and the description of specific service is expressed by inheriting the aspects of the service.

For example, the category of the service profile can be referred to the *UNSPSC*[1] (United Nations Standard Products and Services Code), which is an open, global electronic commerce standard that provides a logical framework for classifying goods and services. The *UNSPSC* is a hierarchical classification, having five levels (segment, family, class, commodity and business function). Each level contains a two-character numerical value and a textual description. For example, the commodity "*TravelAgencies*" (the fourth level) is part of a larger class of services, "*TravelAgents*", which in turn is part of a family of services, "*TravelFacilitation*", which is itself part of segment of services, "*TravelAndFoodAndLodgingAndEntertainmentServices*". The fifth level can be further extended. Here we extend "*EuropeTravel*", "*AsiaTravel*" etc, which are parts of a commodity of services "*TravelAgencies*". The hierarchical classification can be expressed as concept inclusion axioms in TBox:

[1]http://www.unspsc.org

EuropeTravel $\sqsubseteq$ *TravelAgencies, AsiaTravel* $\sqsubseteq$ *TravelAgencies,*

TravelAgencies $\sqsubseteq$ *TravelAgents, TravelAgents* $\sqsubseteq$ *TravelFacilitation,*

TravelFacilitation $\sqsubseteq$ *TravelAndFoodAndLodgingAndEntertainmentServices*

For *EuroTravel* service, the domain knowledge is as follows: *EuroCountry* is a sub-concept of the *Country*, *austria*, *italy*, *germany*, *france* etc are instances of *EuroCountry*. The *VisaCard* and *MasterCard* are disjoint and they are both a kind of *BankCard*.

EuroCountry $\sqsubseteq$ *Country,*

EuroCountry(*austria*), *EuroCountry*(*italy*), *EuroCountry*(*germany*), *EuroCountry*(*france*), ...

VisaCard $\sqsubseteq \neg$ *MasterCard, VisaCard* $\sqsubseteq$ *BankCard, MasterCard* $\sqsubseteq$ *BankCard, ...*

And the aspects of the *EuroTravel* services are as follows: *EuroTravel* has category of *EuropeTravel,* and the service cost is between 300 and 500.

B_{ET}::=*Basic*$\sqcap$($\exists$*hasCategory.EuropeTravel*), Q_{ET}::=*Qos*$\sqcap$ ($\exists$*cost.*$\geq_{300}$) $\sqcap$($\exists$*cost.*$\leq_{500}$),

P_{ET}::=*Precondition*$\sqcap$($\forall$*origin.*$\{$*austria,italy*$\}$)$\sqcap$($\forall$*destination.*$\{$*austria,italy*$\}$)$\sqcap$($\exists$*hasCard.VisaCard*),

I_{ET}::=*Input*$\sqcap$ ($\exists$ *origin.EuroCountry*)$\sqcap$($\exists$ *destination. EuroCountry*)$\sqcap$($\exists$ *price.*$\top_{\mathcal{D}}$) $\sqcap$ ($\exists$*hasCard. VisaCard*),

O_{ET}::=*Output*$\sqcap$($\exists$ *hasConfirmation.Confirmation*), E_{ET}::=*Effect*$\sqcap$ ($\exists$ *hasCharge.Charge*)

Q_{ET} is defined by an expression of the form *Qos*$\sqcap$ ($\exists$*cost.*$\geq_{300}$) $\sqcap$($\exists$*cost.*$\leq_{500}$), here $\geq_{300}$ stands for the unary predicate $\{$n | n$\geq$300$\}$ of all nonnegative integers greater than or equal to 300.

3.2. Reasoning about matchmaking of service

There are degrees of web service like *exact*, *plugin*, *subsume* and *fail*. The subsumption is basic TBox reasoning in a DL system, and others reasoning can be deduced to it. According to Corollary 2.1, we propose an algorithm *WebServiceRelDecision*, which makes the service matching by using subsumption reasoning. Let K=(T, A) be the knowledge base, P and R are service class of provider and that of requester respectively, and *Basic*, *Qos* and *IOPEs* of P are written as $B_P, Q_P, P_P, I_P$and O_P, the respective aspects of R are as B_R, Q_R, P_R, I_R and O_R.

TYPE TMatchDegree = ENUMERATION
(exact, plugIn, subsume,fail)
END

The data structure *TMatchDegree* denotes a degree of service match following definition 2.6. According to Corollary 2.1, if $(B_P \sqsupseteq B_R)$ $\wedge(Q_P \sqsupseteq Q_R)$ $\wedge(P_P \sqsupseteq P_R)$ $\wedge$ $(I_P \sqsupseteq I_R)$ $\wedge$ $(O_P \sqsubseteq O_R)$ $\wedge$ $(E_P \sqsubseteq E_R)$ is satisfied, then we can infer that P is compatible with R (see line 1-2). If different direction is satisfied, then we can get that R is compatible with P (see line 3-4). Lastly, the match degree is achieved (see line 5-10).

The function *subsume*(x, y) is used for checking whether concept x subsumes concept y, which is a basic TBox reasoning in DL. So, the decision of service match degree is reduced to the concept subsumption reasoning.

According to the definition of service matchmaking (definition 2.4), its process is expressed in *WebServiceMatchReason*(). The input R and α[n] are requester and the set of providers in the repository. The output MatchServicesSet is the set of providers which are matched.

Algorithm 1: WebServiceRelDecision

```
Input: $P, R$ are service descriptions of provider and requester, $K$ is knowledge-base
Output: $matchDegree$ is degree of the service match
1  $bSsm = subsume(B_P, B_R), qSsm = subsume(Q_P, Q_R), pSsm = subsume(P_P, P_R), iSsm = subsume(I_P, I_R), oSsm = subsume(O_R, O_P), eSsm = subsume(E_R, E_P)$;
2  $isCptb1 = (bSsm \wedge qSsm \wedge pSsm \wedge iSsm \wedge oSsm \wedge eSsm)$;
3  $bSsm = subsume(B_R, B_P), qSsm = subsume(Q_R, Q_P), pSsm = subsume(P_R, P_P), iSsm = subsume(I_R, I_P), oSsm = subsume(O_P, O_R), eSsm = subsume(E_P, E_R)$;
4  $isCptb2 = (bSsm \wedge qSsm \wedge pSsm \wedge iSsm \wedge oSsm \wedge eSsm)$;
5  switch () do
6      case ($isCptb1 \wedge isCptb2$)   $matchDegree = exact$;
7      case ($isCptb1 \wedge \neg isCptb2$)   $matchDegree = plugIn$;
8      case ($\neg isCptb1 \wedge isCptb2$)   $matchDegree = subsume$;
9      case ($\neg isCptb1 \wedge \neg isCptb2$)   $matchDegree = fail$;
10 end
```

Algorithm 2: WebServiceMatchReason

```
Input: $R, \alpha[n], K$
Output: $MatchServicesSet$ is the set of the services matched
1  for ($i = 1; i \leq sizeof(\alpha[n])$) do
2      $matchDegree = \underline{WebServiceRelDecision}(\alpha[i], R, K)$;
3      if ($matchDegree \in \{exact, plugIn\}$) then
4          $matchItem = (\alpha[i], R, matchDegree)$;
5          $MatchServicesSet = MatchServicesSet + \{matchItem\}$ ;
6      end
7  end
8  $Sort(MatchServicesSet)$;
```

```
TYPE TMatchItem = RECORD
    id : String;
    P, R : String;
    matchDegree : TMatchDegree;
END
```

The data structure *TMatchItem* denotes an item of service match correspondence. For each provider, check whether it is compatible with R (line 2). If the degree is *exact* or *plugin*, then add the match item into set of services matched (line 3-6). Lastly, sort the *MatchServicesSet* by match degree (line 8).

4. Case study

RacerPro is selected as our DL system. RacerPro is a knowledge representation system that implements a highly optimized tableau calculus for a very expressive description logic. It offers reasoning services for multiple TBoxes and for multiple ABoxes as well. The system implements the description logic ALCQHI$_{R+}$.

RacerPro works in a client/server model and the server is RacerPro reasoner (Ver1.9.0)[2], education license. The reasoning about service matchmaking is implemented by using JRacer (Ver 1.8)[3], which is RacerPro's API in Java, and provides a simple way to communicate with a RacerPro server based on TCP sockets. For example, the function *subsume*(x, y) in *WebServiceRelDecision*( ) can be achieved by query (concept-subsumes? x y) in RacerPro.

Let the *EuroTravel* service be requester R, and let *SP*$_1$, *SP*$_2$, *SP*$_3$, *SP*$_4$, *SP*$_5$ be a group of service providers in repository. In contrast with R, *SP*$_1$ requires as a precondition destination in Europe country, and as input the visa card or master card; *SP*$_2$ requires as basic information category in *TravelAgencies*, and as a precondition card in bank card, and as input origin and destination in country; *SP*$_3$ requires as a precondition destination in Asian country; *SP*$_4$ requires as a precondition origin in *austria* and destination in *italy*; *SP*$_5$ generates an email confirmation and discount as output. The descriptions of service *SP*$_i$, i={1,2,3,4,5}, are expressed as follows.

SP$_i$::=($\exists$*hasProfile.Profile*$_i$)$\sqcap$($\exists$*hasCapability.Capability*$_i$), *Profile*$_i$::=($\exists$*hasBasic.*B_i)$\sqcap$($\exists$*hasQos.*Q_i), *Capability*$_i$::= ($\exists$*hasPrecondition.*P_i)$\sqcap$ ($\exists$*hasInput.*I_i)$\sqcap$ ($\exists$*hasOutput.*O_i)$\sqcap$ ($\exists$*hasEffect.*E_i) .

The aspects of service *SP*$_i$, such as B_i, Q_i, P_i, I_i, O_i and E_i, are expressed as follows:

B_1::=*Basic*$\sqcap$($\exists$*hasCategory.EuropeTravel*),

Q_1::=*Qos*$\sqcap$($\exists$*cost.*$\geq_{300}$) $\sqcap$($\exists$*cost.*$\leq_{500}$),

P_1::=*Precondition*$\sqcap$($\forall$*destination.EuroCountry*)$\sqcap$($\exists$*hasCard.*(*VisaCard*$\sqcup$*MasterCard*)),

I_1::=*Input*$\sqcap$($\exists$ *origin. Country*) $\sqcap$ ($\exists$ *destination. EuroCountry*) $\sqcap$ ($\exists$ *price.*$\top_{\mathcal{D}}$) $\sqcap$ ($\exists$*hasCard.* (*VisaCard*$\sqcup$*MasterCard*)),

O_1::=*Output*$\sqcap$ ($\exists$*hasConfirmation.Confirmation*),

E_1::=*Effect*$\sqcap$ ($\exists$ *hasCharge.Charge*)

B_2::=*Basic*$\sqcap$ ($\exists$*hasCategory.TravelAgencies*),

Q_2::=*Qos*$\sqcap$ ($\exists$*cost.*$\geq_{300}$) $\sqcap$($\exists$*cost.*$\leq_{500}$),

P_2::=*Precondition*$\sqcap$ ($\exists$*hasCard.BankCard*),

I$_2$::=*Input*$\sqcap$ ($\exists$ *origin. Country*) $\sqcap$ ($\exists$ *destination. Country*) $\sqcap$ ($\exists$ *price.*$\top_{\mathcal{D}}$) $\sqcap$ ($\exists$*hasCard.BankCard*),

O_2::=*Output*$\sqcap$ ($\exists$*hasConfirmation.Confirmation*) ,

E_2::=*Effect*$\sqcap$ ($\exists$ *hasCharge.Charge*)

B_3::=*Basic*$\sqcap$($\exists$*hasCategory.AsiaTravel*),

Q_3::=*Qos*$\sqcap$ ($\exists$*cost.*$\geq_{300}$) $\sqcap$($\exists$*cost.*$\leq_{500}$),

P_3::=*Precondition*$\sqcap$ ($\forall$*destination.AsianCountry*) $\sqcap$ ($\exists$*hasCard.VisaCard*),

I_3::=*Input*$\sqcap$ ($\exists$ *origin. Country*) $\sqcap$ ($\exists$ *destination. AsianCountry*) $\sqcap$ ($\exists$ *price.*$\top_{\mathcal{D}}$)$\sqcap$ ($\exists$*hasCard. VisaCard*),

O_3::=*Output*$\sqcap$ ($\exists$*hasConfirmation.Confirmation*) ,

E_3::=*Effect*$\sqcap$ ($\exists$ *hasCharge.Charge*)

[2]http://www.racer-systems.com/

[3]http://www.racer-systems.com/products/download/nativelibraries.phtml

B_4::=*Basic*$\sqcap$ ($\exists$*hasCategory.EuropeTravel*),

Q_4::=*Qos*$\sqcap$ ($\exists$*cost.*$\geq_{300}$) $\sqcap$ ($\exists$*cost.*$\leq_{500}$),

P_4::=*Precondition*$\sqcap$($\forall$*origin.*$\{$*austria*$\}$)$\sqcap$($\forall$*destination.*$\{$*italy*$\}$)$\sqcap$ ($\exists$*hasCard.VisaCard*),

I_4::=*Input*$\sqcap$ ($\exists$ *origin. EuroCountry*) $\sqcap$ ($\exists$ *destination. EuroCountry*) $\sqcap$ ($\exists$ *price.*$\top_{\mathcal{D}}$)$\sqcap$ ($\exists$*hasCard. VisaCard*),

O_4::=*Output*$\sqcap$ ($\exists$*hasConfirmation.Confirmation*) ,

E_4::=*Effect*$\sqcap$ ($\exists$ *hasCharge.Charge*)

B_5::=*Basic*$\sqcap$ ($\exists$*hasCategory.EuropeTravel*),

Q_5::=*Qos*$\sqcap$ ($\exists$*cost.*$\geq_{300}$) $\sqcap$($\exists$*cost.*$\leq_{500}$),

P_5::=*Precondition*$\sqcap$($\forall$*origin.*$\{$*austria,italy*$\}$)$\sqcap$($\forall$*destination.*$\{$*austria,italy*$\}$)$\sqcap$ ($\exists$*hasCard.VisaCard*),

I_5::=*Input*$\sqcap$($\exists$ *origin. EuroCountry*) $\sqcap$ ($\exists$ *destination. EuroCountry*) $\sqcap$ ($\exists$ *price.*$\top_{\mathcal{D}}$)$\sqcap$ ($\exists$*hasCard. VisaCard*),

O_5::=*Output*$\sqcap$ ($\exists$*hasConfirmation. EmailConfirmation*) $\sqcap$ ($\exists$*hasPromotion.Discount*) ,

E_5::=*Effect*$\sqcap$ ($\exists$ *hasCharge.Charge*)

The domain knowledge is expressed in knowledge base as:

EuroCountry $\sqsubseteq$ *Country*, *AsianCountry* $\sqsubseteq$ *Country*, *AsianCountry* $\sqsubseteq$ $\neg$*EuroCountry*, *VisaCard* $\sqsubseteq$ $\neg$ *MasterCard*, *VisaCard* $\sqsubseteq$ *BankCard*, *MasterCard* $\sqsubseteq$ *BankCard*, *EmailConfirmation* $\sqsubseteq$ *Confirmation*

All the top-level concepts of the semantic web service meta-model, and the description of specific service class (e.g. the EuroTravel service) and relative domain knowledge are expressed in RacerPro knowledge base, the key part can be seen in Appendix A. And the algorithm *WebServiceRelDecision*() can be implemented by the query inference in RacerPro (see the bottom section in AppendixA). The results of match are shown in Table1.

Table 1. Match results

	B	Q	P	I	O	E	match degree
SP1	B=B1	Q=Q1	$P \sqsubseteq P1$	$I \sqsubseteq I1$	O=O1	E=E1	plugIn
SP2	$B \sqsubseteq B2$	Q=Q2	$P \sqsubseteq P2$	$I \sqsubseteq I2$	O=O2	E=E2	plugIn
SP3	$B \sqcap B3 \neq \perp$	$Q \sqcap Q3 \neq \perp$	$P \sqcap P3 \neq \perp$	$I \sqcap I3 \neq \perp$	O=O3	E=E3	fail
SP4	B=B4	Q=Q4	$P \sqsupseteq P4$	I=I4	O=O4	E=E4	subsume
SP5	B=B5	Q=Q5	P=P5	I=I5	$O \sqsupseteq O5$	E=E5	plugIn

In contrast with service requester R, if the service candidate *SP*$_i$ have less and general basic/Qos/ precondition/input parameters, and generate more and specific output parameters, then *SP*$_i$ can have more opportunities to be compatible with R. If the aspects of service are expressed precisely, then we can get really good match results.

The result of the research effort shows that web services can indeed find each other automatically and interoperate autonomously without the need of hardcoded interactions. Our matching algorithm provides a way for automatic dynamic discovery, selection of web services, which is a crucial feature in the web of the future in which services dynamically reconfigure their supply chain to better match changes in the market.

5. Related work

For the string based service match, UDDI is a method of matching services in terms of key words. And Müller[11] described the service as the main service properties and the service comparing strategy is based on string matching.

For the logic based service match, Gonzalez-Castillo[12] constructed the subsumption tree of the serviceDescription, and the tree node is the concept of the serviceDescription. This approach matches the service by finding the equivalent concepts, sub-concepts and super-concepts. This approach does not differentiate between input and output.

There are many service matchmaking approaches[4, 5, 6, 7, 8], which are based on existing semantic service models, such as DAML-S/OWL-S and WSMO.

Paolucci[5] proposed the service matchmaking based on DAML-S, the match algorithm is intuitive: for each output of the requester, there is a matching output in the provider, and the matching between inputs is computed following the same algorithm but reversed order. The approach observes ”satisfy-direction difference” (see theorem 2.1), but only inputs/outputs are considered. Li [6] expressed the input and output of service as concept *ServiceProfile*, and classified the input/output parts by using Racer system. Then, the service matchmaking is achieved by computing the input/output parts of requester’s subsumption relationships w.r.t. the input/output parts of all the provider ServiceProfiles. Its algorithm *doMatch*() does not satisfy the ”satisfy-direction difference” and only input and output parts are considered. Klusch[7] proposed a hybrid semantic Web service matching that exploits both explicit and implicit semantics. Besides explicit semantics, it complements logic based reasoning with approximate matching based on syntactic IR based similarity computations. While the explicit semantics include only input and output. Dong[13] proposed the matching approach, which took into account IOPEs. Inputs and outputs (IO) match followed the match algorithm in [5], and preconditions and effects (PE) are modeled in DL ABox.

Ambroszkiewicz[14] proposed an approach called enTish, in which the services are composed on the fly in order to realize clients’ requests. enTish proposed a well defined logic language called Entish, and the request is expressed as the formula. It is open, and is of distributed use to enable uses to introduce new resource type, functions and relations with URIs.

Meanwhile, the most of current semantic service discovery methods perform service I/O based profile matching, there exists no matchmaker that performs an integrated service matching by additional reasoning on logically defined preconditions, effects, Qos and so on. Our services description model describes the various aspects of the web services as concepts in TBox of DL knowledge base, which enables the service matchmaking by DL reasoning.

Conceptual models with DL offer more expressive facilities for modeling, such as model checking[15] and data mining[16].

6. Conclusion

We proposed a formal model in DL for the representation of semantic web services, which facilitates the service matchmaking. The aspects of the service (such as basic, Qos, IOPEs) are expressed as concepts in TBox. The service compatibility theorem is proposed, which promotes the service match algorithm.

The algorithm can be used to reason about the service matchmaking by using DL reasoner (such as RacerPro).

The main features of this approach can be summarized as follows:

(1) The semantic service model based on DL focuses on the service capabilities, so it is abstract, simple and easy to model for domain experts.

(2) The semantic technology makes the model machine-processable, which enables the service requester and provider ”know” each other and leads to high precision for service matchmaking.

(3) The model introduces all aspects of the service, on which the service match algorithm depends, while many other approaches only consider input and output.

(4) The service aspects are expressed as concepts in DL knowledge base, so the services matchmaking is transformed into the match of concepts by using TBox reasoning.

(5) The DL system promises the correctness of the service matchmaking, if service capabilities are described precisely.

Our model is mainly used to describe the static aspects of the services. Future work includes modeling the dynamic behavior of the services, which concentrates on the state of the world.

References

[1] Booth D, Haas H, McCabe F, et al: Web Services Architecture. W3C Working Group Note, 11 February 2004. http://www.w3.org/TR/ws-arch/

[2] Martin D, Burstein M, Hobbs J, et al : OWL-S: Semantic Markup for Web Services, 2004, http://www.w3.org/Submission/OWL-S/, 2004-11-22

[3] Roman D, Keller U, Lausen H, et al : Web Service Modeling Ontology[J]. *Applied Ontology*, 2005, 1(1): 77–106.

[4] Sycara K, Paolucci M, Ankolekar A, et al : Automated discovery, interaction and composition of Semantic Web services[J]. *Journal of Web Semantics*,2003, 1(1):27–46.

[5] Paolucci M, Kawamura T, Payne TR, Sycara K : Semantic matching of Web services capabilities[A]. In: Proc. of the Int’l Semantic Web Conf. (ISWC)[C]. LNCS 2342, 2002, 333–347.

[6] Li L, Horrocks I: A Software Framework for Matchmaking Based on Semantic Web Technology[J]. *International Journal of Electronic Commerce*, 2004, 8(4):39–60.

[7] Klusch M, Fries B, Sycara K: Automated Semantic Web Service Discovery with OWLS-MX[A]. In: Proc. of the fifth international joint conference on Autonomous agents and multiagent systems[C], 2006,915–922.

[8] Keller U, Lara R, Lausen H, et al : Automatic Location of Services[A]. In: Proc. of the 2nd European Semantic Web Conference[C], LNCS 3532, 2005.

[9] Baader F, Calvanese D, McGuinness D, et al: The description logic handbook: theory, implementations and applications[M]. Cambridge University Press, 2003.

[10] Yao Y, Su S, Yang F: Service Matching Based on Semantic Descriptions[A]. In: Proc. of the Advanced Int’l Conf. on Telecommunications and Int’l Conf. on Internet and Web Applications and Services(AICT/ICIW 2006) [C], 2006.

[11] Müller H, Hilbrich T and Kühnel R: An Assistant Agent[J]. *Fundamenta Informaticae*, 1999, 39(3): 327–336.

[12] Gonzalez-Castillo J, Trastour D, Bartolini C: Description logics for matchmaking of services[A]. In: KI-2001 Workshop on Applications of Description Logics[C], 2001.

[13] Dong T, Li Q, Zhang K, et al : An Extended Matching Method for Semantic Web Service in Collaboration Environment[A]. In: Proc. of the 2007 11th International Conference on Computer Supported Cooperative Work in Design[C].2007,508–513

[14] Ambroszkiewicz S: enTish: An Approach to Service Description and Composition[M]. Instytut Podstaw Informatyki Polskiej Akademii Nauk, Warszawa 2003

[15] Zhao X, Huang Z. A Formal Framework for Reasoning on Metadata based on CWM[A]. In: Proc. of 25th International Conference on Conceptual Modeling(ER 2006) [C]. LNCS 4215, Springer, 2006, 371–384.

[16] Zhu X, Huang Z, Conceptual modeling rules extracting for data streams[J], Knowledge-based System. Elsevier Press, 2008, 21(8): 934–940.

A. Semantic description of services and reason about matchmaking in RacerPro

```
;; (note that the lines begin with ";;" are comments)
    ;;———— semantic web service meta-model ————
(implies Basic WSAspect)
(implies Precondition WSAspect)
(implies Input WSAspect)
(implies Output WSAspect)
(implies Effect WSAspect)...
    ;; ———— domain knowledge ————
(implies EuroCountry Country )
(implies AsianCountry Country)
(disjoint EuroCountry MasterCard)
(implies EmailConfirmation Confirmation)
(instance austria EuroCountry)
(instance italy EuroCountry)...
    ;;———— requester R ————
(define-concept B (and Basic (some hasCategory EuropeTravel)))
(define-concept Q (and Qos (>= cost 300) (<= cost 500)))
(define-concept P (and Precondition (all origin (one-of austria italy))(all destination (one-of austria italy))
(some hasCard VisaCard)))
(define-concept I (and Input (some origin EuroCountry)(some destination EuroCountry) (a price)))
(define-concept O (and Output (some hasConfirmation Confirmation)))
(define-concept E (and Effect (some hasCharge Charge)))
```

```
    ;; ————— SP1 —————
(define-concept B1 (and Basic (some hasCategory EuropeTravel)))
(define-concept Q1 (and Qos (>= cost 300) (<= cost 500)))
(define-concept P1 (and Precondition (all destination EuroCountry) (some hasCard (or VisaCard Mas-
terCard))))
(define-concept I1 (and Input (some origin Country)(some destination EuroCountry) (a price)))
(define-concept O1 (and Output (some hasConfirmation Confirmation)))
(define-concept E1 (and Effect (some hasCharge Charge)))...
    ;;————— campatible(SP1, R)?—————
(concept-subsumes? B1 B)
(concept-subsumes? Q1 Q)
(concept-subsumes? P1 P)
(concept-subsumes? I1 I)
(concept-subsumes? O O1)
(concept-subsumes? E E1)...
```