Semantic Interpretation and Matching of Web Services

Chang Xu, Shing-Chi Cheung, and Xiangye Xiao

Department of Computer Science
Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
{changxu,scc,xiaoxy}@cs.ust.hk

Abstract. A major issue in the study of semantic Web services concerns the matching problem of Web services. Various techniques for this problem have been proposed. Typical ones include *FSM* modeling, *DAML-S* ontology matching, description logics reasoning, and *WSDL* dual operation composition. They often assume the availability of concept semantic relations, based on which the capability satisfiability is evaluated. However, we find that the use of semantic relations alone in the satisfiability evaluation may lead to inappropriate results. In this paper, we study the problem and classify the existing techniques of satisfiability evaluation into three approaches, namely, set inclusion checking, concept coverage comparison and concept subsumption reasoning. Two different semantic interpretations, namely, capacity interpretation and restriction interpretation, are identified. However, each of the three approaches assumes only one interpretation and its evaluation is inapplicable to the other interpretation. To address this limitation, a novel interpretation model, called *CRI* model, is formulated. This model supports both semantic interpretations, and allows the satisfiability evaluation to be uniformly conducted. Finally, we present an algorithm for the unified satisfiability evaluation.

1 Introduction

Much attention has been paid to the study of semantic Web services [5], which aims to explore a better way to describe and implement Web services, making them accessible to automated agents. Ontology plays a key role in the Semantic Web by providing common vocabularies shared by applications. Currently, the Web Services Description Language (*WSDL*) [19] does not support semantic descriptions for Web services [4], and the Universal Description, Discovery and Integration (*UDDI*) [20] also cannot provide adequate documentation of Web service capabilities [4]. To address their limitations, ontology languages (e.g., *DAML+OIL* [15]) have been proposed whose goal is to facilitate the automation of tasks including Web service discovery, execution, composition and interoperation.

Capability matching of Web services is a major research issue in this field. Generally, when an agent is given a service request, it tries to match it against available service advertisements stored in its Web service repository. The matching is conducted in terms of capabilities based on the underlying semantics of the concepts involved. A variety of techniques have been proposed. Popular ones include finite-state machine (*FSM*) modeling [2], *DAML-S* [16] ontology matching [11], Description Logics (*DL*s) [1] reasoning [4], and *WSDL* dual operation composition [6].

A service advertisement is said to be matched for a service request only when the capabilities of this advertisement satisfy the requirements of this request. The match-

P. Atzeni et al. (Eds.): ER 2004, LNCS 3288, pp. 542–554, 2004.

ing process is called *satisfiability evaluation*. We classify existing techniques in the satisfiability evaluation into three approaches:

- **Set Inclusion Checking:** In this approach, each set contains available resources or required resources. The goal of checking is to find inclusion relations between sets. For example, if B_1 = {"*SOAP*"} [18] and B_2 = {"*SOAP*", "*HTTP*"}, B_2 is said to be able to satisfy B_1 [6].
- **Concept Coverage Comparing:** This approach uses concepts to represent capabilities or requirements. In comparison, a more general concept can satisfy a more specific concept. For example, concept *Vehicle* satisfies concept *Car* [11].
- **Concept Subsumption Reasoning:** In this approach, capabilities or requirements are represented in *DL* clauses. In reasoning, a more specific *DL* clause can satisfy a more general *DL* clause. For example, $\forall item.(PC \cap \geq_{256} memorySize)$ satisfies $\forall item.(PC \cap \geq_{128} memorySize)$ [4].

The first two approaches are essentially equivalent. To illustrate this, we assume that *Vehicle* has two sub-concepts: *Car* and *Bus*. After we rewrite *Vehicle* as {*Car*, *Bus*} and *Car* as {*Car*}, *Vehicle* satisfies *Car* iff {*Car*, *Bus*} includes {*Car*}. Next, the last two approaches have opposite criteria for the satisfiability evaluation. The reason is that each *DL* clause can be regarded as a concept. Finally, all the three approaches make use of semantic relations between concepts.

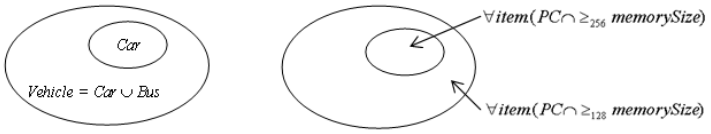

Fig. 1. The *Vehicle* and *PC* Examples

However, the use of concept semantic relations alone in the satisfiability evaluation may lead to inappropriate results. It is because a concept can be subject to multiple interpretations under different contexts. The *Vehicle* example actually adopts the *capacity interpretation* of *Vehicle* that refers to the semantics of *Vehicle* and all its sub-concepts. *Vehicle* can satisfy *Car* under this context. But the *PC* example adopts *restriction interpretations* of two *DL* clauses: $\forall item.(PC \cap \geq_{256} memorySize)$ and $\forall item.(PC \cap \geq_{128} memorySize)$. Under this context, the former is more restrictive than the latter. Therefore, the requirement of a *PC* with memory larger than 128MB can be satisfied by the offering of a *PC* with memory larger than 256MB (Fig. 1).

The above two examples indicates that an offering labeled by a more general concept does not necessarily satisfy a requirement labeled by a more specific concept, and vice versa. The satisfiability evaluation should depend on how the involved concepts are semantically interpreted. We call this *semantic interpretation* which can be based on either capacity or restriction. More rigorous definitions for these interpretations will be given later.

Each of the three approaches discussed actually assumes only one semantic interpretation, and its satisfiability evaluation is inapplicable to the other semantic interpretation. To address this limitation, we propose a unified capability matching model for Web services, in which the satisfiability evaluation can be performed uniformly based on two semantic interpretations.

The remainder of this paper is organized as follows. Section 2 introduces related work. Section 3 proposes our solution to the Web service capability matching problem. Section 4 discusses how to implement the capability matching system. Section 5 evaluates our system using four well-known criteria, and presents our two extra features. In the last section, we conclude our contributions and explore future work.

2 Related Work

Various studies have been conducted on the service capability matching issue. Gao et al. [2] formally defined *exact match* and *plug-in match* for Web service capabilities based on abstract *FSM* models. They also proposed a capability description language *SCDL* and a solution for comparing service capabilities. However, the work focuses mainly on signature matching rather than semantic matching. Moreover, it is difficult to use the *FSM* approach to precisely describe Web service capabilities.

Paolucci et al. [11] suggested semantic matching for Web services and proposed a solution based on *DAML-S*. The work presents a matching framework to examine every advertisement and request pair. It also suggests flexible matching by providing four matching degrees based on a predefined taxonomy tree. However, the work only considers capacity interpretations of concepts. Furthermore, the technique addresses mainly *is-a* relations between concepts.

Li and Horrocks [4] described a service matchmaking prototype using a *DAML-S* ontology and a *DL* reasoner. This approach is useful to describe the semantics of service adverts and queries. However, each *DL* expression can be regarded as a conjunction of several restriction clauses. If a given advert and query contain different restriction clauses at some corresponding position, the algorithm cannot work well. To address this problem, they proposed to minimize the *DL* expression under examination by moving unimportant *providedBy* and *requestedBy* clauses out of the *profile* that participates in comparison. This adjustment only partially solves the problem because the remaining part of the *profile* may also include some information that will affect the matching result. Another limitation of the work is that the authoring of *DL* expressions requires users' familiarity of formal logics.

Medjahed et al. [6] proposed an ontology-based framework for Web service composition. Several composability rules are deployed to compare the syntactic and semantic features of Web services. A composability model that covers from low-level binding mode to high-level composition soundness was formulated. In the satisfiability evaluation, the framework only adopts simple set inclusion checking. Essentially, the approach is based on the capacity interpretations of concepts.

Other related work includes *LARKS* [12] and *ARLAS* [10] matchmaking systems. *LARKS* defines five techniques for service matchmaking. Those techniques mostly compare service text descriptions, signatures, and logical constraints about inputs and outputs [6]. *ATLAS* defines two methods to compare services for functional attributes and capabilities [6].

Generally, existing techniques decide matching results based on relations between compared concept pairs. In our approach, these relations are used as the intermediate result in the satisfiability evaluation. To accurately perform capability matching, we take into account the semantic interpretations of concepts involved. Our satisfiability evaluation differs from others in that it uniformly performs multiple satisfiability

evaluations based on semantic interpretations of concepts. Therefore, the final matching result is a synthesis of multiple satisfiability evaluations.

3 Semantic Interpretation and Capability Matching

Let us first overview some preliminary concepts on conceptual modeling before presenting our satisfiability evaluation technique, which is based on semantic interpretations of concepts. A capability matching algorithm will be subsequently formulated.

3.1 Preliminary

Definition 1 (Concept-Instance Structure): *A concept-instance (CI) structure CIS = (C, I, cimap) is a triple where:*

- *C is a set of concepts;*
- *I is a set of instances;*
- *Total function cimap:* $C \rightarrow 2^I$ *relates a concept in C with a non-empty set of instances in I.*

Fig. 2 gives the graphical representation of an illustrative example. The following is the corresponding *CI* structure:

$CIS = (C, I, cimap)$
$C = \{Vehicle, Bus, Car, Sedan, SUV\}$
$I = \{b_1, b_2, se_1, se_2, suv\}$
$cimap(Vehicle) = \{b_1, b_2, se_1, se_2, suv\}$
$cimap(Bus) = \{b_1, b_2\}$
$cimap(Car) = \{se_1, se_2, suv\}$
$cimap(Sedan) = \{se_1, se_2\}$
$cimap(SUV) = \{suv\}$

Definition 2 (Equivalent, subsumed, including, intersecting and disjoint): *In a CI structure CIS, any two concepts* c_1 *and* c_2 *are subject to one of the following relations:*

- *equivalent: if* $cimap_{CIS}(c_1) = cimap_{CIS}(c_2)$;
- *subsumed: if* $cimap_{CIS}(c_1) \subset cimap_{CIS}(c_2)$;
- *including: if* $cimap_{CIS}(c_1) \supset cimap_{CIS}(c_2)$;
- *disjoint: if* $cimap_{CIS}(c_1) \cap cimap_{CIS}(c_2) = \phi$;
- *intersecting: otherwise.*

Definition 3 (Descendent concept): *If concepts* c_1 *and* c_2 *have an equivalent or subsumed relation,* c_1 *is said to be a descendent concept of* c_2.

Since the equivalent relation is reflexive, a concept is a descendent concept of itself. Tables 1 and 2 give the concept relations and the descendent concepts, respectively, for the example of Fig. 2.

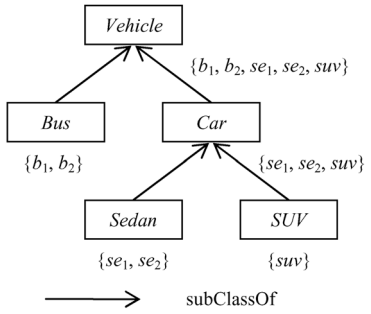

Fig. 2. An illustrative example

Table 1. The concept relation table (c_1 to c_2)

c_1 \ c_2	*Vehicle*	*Bus*	*Car*	*Sedan*	*SUV*
Vehicle	equivalent	including	including	including	including
Bus	subsumed	equivalent	disjoint	disjoint	disjoint
Car	subsumed	disjoint	equivalent	including	including
Sedan	subsumed	disjoint	subsumed	equivalent	disjoint
SUV	subsumed	disjoint	subsumed	disjoint	equivalent

Table 2. The descendent concept table

For *Vehicle*	*Vehicle*, *Bus*, *Car*, *Sedan*, *SUV*
For *Bus*	*Bus*
For *Car*	*Car*, *Sedan*, *SUV*
For *Sedan*	*Sedan*
For *SUV*	*SUV*

Definition 4 (Coverage): *Suppose that $c_1,\ldots, c_n$ are concepts in a CI structure CIS. If I is a set of instances that satisfies $I \subseteq cimap_{CIS}(c_1) \cup \ldots \cup cimap_{CIS}(c_n)$ and $(\exists e \in cimap_{CIS}(c_1), e \in I) \wedge \ldots \wedge (\exists e \in cimap_{CIS}(c_n), e \in I)$, I is said to have a coverage relation with $\{c_1, \ldots, c_n\}$, written as: $I\ R_{cov}\ \{c_1, \ldots, c_n\}$.*

If I has a coverage relation with $\{c_1, \ldots, c_n\}$, it means that I is subject to some relation with each concept in set $\{c_1, \ldots, c_n\}$. Some coverage relations for the example in Fig. 2 are given below:

$\{b_1\}\ R_{cov}\ \{BUS\}$ $\quad$ $\{b_2\}\ R_{cov}\ \{BUS\}$ $\quad$ $\{b_1, b_2\}\ R_{cov}\ \{BUS\}$
$\{se_1, suv\}\ R_{cov}\ \{Sedan, SUV\}$ $\quad$ $\{se_2, suv\}\ R_{cov}\ \{Sedan, SUV\}$
$\{se_1, se_2, suv\}\ R_{cov}\ \{Sedan, SUV\}$

3.2 Satisfiability Evaluation

3.2.1 Semantic Interpretation

Definition 5 (Restriction interpretation): *If c is a concept in a CI structure CIS, the restriction interpretation $\vartheta^R_{CIS}(c)$ of c is the set of all sets that has a coverage relation with $\{c\}$: $\vartheta^R_{CIS}(c) = \{I \mid I\ R_{cov}\ \{c\}\}$.*

Definition 6 (Capacity interpretation): *If c is a concept in a CI structure CIS, the capacity interpretation $\vartheta^C_{CIS}(c)$ of c is the set of all sets that has a coverage relation with the set of all descendent concepts of c: $\vartheta^C_{CIS}(c) = \{I \mid I\ R_{cov}\ \{c_1,\ldots,c_n\}$, $c_1,\ldots,c_n$ are all descendant concepts of $c\}$.*

The restriction interpretation of a concept represents the semantics of this concept, while the capacity interpretation of a concept represents the semantics of all descendent concepts of this concept. Some restriction interpretations and capacity interpretations for the example in Fig. 2 are given below:

$\vartheta^R_{CIS}(Sedan) = \{\{se_1\}, \{se_2\}, \{se_1, se_2\}\}$ $\quad$ $\vartheta^C_{CIS}(Sedan) = \{\{se_1\}, \{se_2\}, \{se_1, se_2\}\}$
$\vartheta^R_{CIS}(SUV) = \{\{suv\}\}$ $\quad$ $\vartheta^C_{CIS}(SUV) = \{\{suv\}\}$
$\vartheta^R_{CIS}(Car) = \{\{se_1\}, \{se_2\}, \{suv\}, \{se_1, se_2\}, \{se_1, suv\}, \{se_2, suv\}, \{se_1, se_2, suv\}\}$
$\vartheta^R_{CIS}(Car) = \{\{se_1, suv\}, \{se_2, suv\}, \{se_1, se_2, suv\}\}$

Definition 7 (Semantic interpretation): *An interpretation is a semantic interpretation iff it is a restriction interpretation or a capacity interpretation.*

Semantic interpretation is used for the satisfiability evaluation in Web service capability matching. Each concept has two semantic interpretations: restriction interpretation and capacity interpretation.

3.2.2 Semantics Satisfiability

Definition 8 (Semantics satisfiability): *The evaluation for semantics satisfiability compares the semantic interpretations of concepts. Given two concepts c_1 and c_2 in a CI structure CIS and their semantic interpretations $\vartheta_{CIS}(c_1)$ and $\vartheta_{CIS}(c_2)$,*

- *If* $\forall e_2 \in \vartheta_{CIS}(c_2), \exists e_1 \in \vartheta_{CIS}(c_1), e_1 \subseteq e_2$, $\vartheta_{CIS}(c_1)$ *is said to be satisfied by* $\vartheta_{CIS}(c_2)$.
- *Otherwise,* $\vartheta_{CIS}(c_1)$ *is said not to be satisfied by* $\vartheta_{CIS}(c_2)$.

Definition 8 unifies the impact of two different semantic interpretations so that the satisfiability evaluation can be done uniformly under different contexts. Let us denote $\lhd$ as the "*is satisfied by*" relation. Some semantics satisfiability results for the example in Fig. 2 are given below:

$\vartheta^{C}_{CIS}(Bus) \lhd \vartheta^{C}_{CIS}(Vehicle)$ $\quad$ $\vartheta^{C}_{CIS}(Sedan) \lhd \vartheta^{C}_{CIS}(Car)$

$\vartheta^{R}_{CIS}(Vehicle) \lhd \vartheta^{R}_{CIS}(Bus)$ $\quad$ $\vartheta^{R}_{CIS}(Car) \lhd \vartheta^{R}_{CIS}(Sedan)$

$\vartheta^{C}_{CIS}(Sedan) \lhd \vartheta^{R}_{CIS}(Sedan)$ $\quad$ $\vartheta^{C}_{CIS}(SUV) \lhd \vartheta^{R}_{CIS}(SUV)$

$\vartheta^{R}_{CIS}(Sedan) \lhd \vartheta^{C}_{CIS}(Car)$ $\quad$ $\vartheta^{R}_{CIS}(Car) \lhd \vartheta^{C}_{CIS}(Sedan)$

3.2.3 Satisfiability Result

Given two concepts, the result of the satisfiability evaluation depends on two factors: (i) the relation between them and (ii) the semantic interpretations of them. The following satisfiability result table can be derived from Definition 8:

Table 3. The result of satisfiability evaluation of $\vartheta_{CIS}(c_1)$ and $\vartheta_{CIS}(c_2)$

Relation (c_1 and c_2)	equivalent			subsumed			including			intersecting			disjoint		
Figure															
Satisfiability result	c_1 \ c_2	R	C	c_1 \ c_2	R	C	c_1 \ c_2	R	C	c_1 \ c_2	R	C	c_1 \ c_2	R	C
	R	√	√	R	×	√	R	√	√	R	×	$*_2$	R	×	×
	C	$*_1$	√	C	×	√	C	×	×	C	×	×	C	×	×

Notes:

- R: restriction interpretation; C: capacity interpretation.
- √: $\vartheta_{CIS}(c_1) \lhd \vartheta_{CIS}(c_2)$; ×: $\neg(\vartheta_{CIS}(c_1) \lhd \vartheta_{CIS}(c_2))$.
- $*_1$: $\vartheta_{CIS}(c_1) \lhd \vartheta_{CIS}(c_2)$ iff all descendent concepts of c_1 and c_2 are equivalent; $*_2$: $\vartheta_{CIS}(c_1) \lhd \vartheta_{CIS}(c_2)$ iff c_1 has descendent concepts other than itself and each descendent concept c satisfies $cimap_{CIS}(c) \cap cimap_{CIS}(c_1) \cap cimap_{CIS}(c_2) \neq \phi$.

3.2.4 Satisfiability Theorems and Transitive Law

To effectively use the satisfiability result table in Web service capability matching, we generalize the following four satisfiability theorems and a transitive law:

Theorem 1. *The restriction interpretation / capacity interpretation of a concept can satisfy the restriction interpretation / capacity interpretation of any concept equivalent to this concept.*

Theorem 2. *The capacity interpretation of a concept can satisfy the restriction interpretation of any concept equivalent to this concept.*

Theorem 3. *The capacity interpretation of a concept can satisfy the capacity interpretation of any descendent concept of itself.*

Theorem 4. *The restriction interpretation of a concept can be satisfied by the restriction interpretation of any descendent concept of itself.*

Theorem 5. *Transitive law applies to semantics satisfiability.*

Theorems 1 and 2 concern the equivalent part of Table 3, while theorems 3 and 4 concern the subsumed part and the including part of the table, respectively. The proof of Theorem 1 is naturally followed from the definitions of capacity and restriction interpretations. The proofs for the other four theorems can be found in [13].

3.2.5 Satisfiability Evaluation

Suppose that $c_1, \ldots, c_n$ are concepts in a CI structure *CIS*, such that c_n is a descendent concept of $c_{n-1}, \ldots$, and c_2 is a descendent concept of c_1. Note that c_n has no descendent concept other than itself. We have two satisfiability sequences: $\vartheta^{R}_{CIS}(c_1) \lhd \ldots \lhd \vartheta^{R}_{CIS}(c_n)$, and $\vartheta^{C}_{CIS}(c_n) \lhd \ldots \lhd \vartheta^{C}_{CIS}(c_1)$, where $\vartheta^{R}_{CIS}(c_n) = \vartheta^{C}_{CIS}(c_n)$ (Fig. 3).

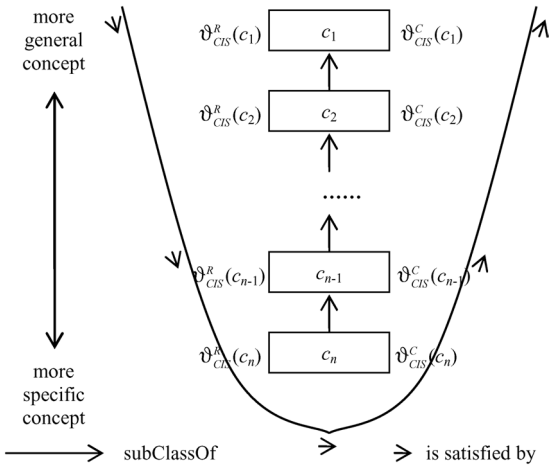

Fig. 3. A satisfiability sequence

The first sequence gives the satisfiability evaluation result between restriction interpretations. A more specific concept can satisfy a more general concept under this semantic interpretation. It is what the concept subsumption reasoning approach is to achieve. The second sequence gives satisfiability evaluation result between capacity interpretations. A more general concept can satisfy a more specific concept under this semantic interpretation. It is what the concept coverage comparing approach is to achieve.

Our satisfiability evaluation generalizes the above two approaches by supporting both restriction and capacity interinterpretations. In addition, it allows dynamic specification of semantic interpretations. This feature is useuseful. Let us assume a binding protocol ontology in Fig. 4. A service provider, which supports

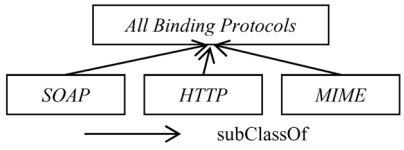

Fig. 4. A binding protocol ontology

SOAP, *HTTP* and *MIME,* advertises his protocols as *All Binding Protocols* using a capacity interpretation. Users looking for a service supporting all binding protocols can make a request for *All Binding Protocols* using a capacity interpretation. However, if users are happy with any binding protocol, they can select a restriction interpretation instead. Dynamic specification of semantic interpretations enables service providers and users to flexibly express their capabilities and requirements.

3.3 CRI Models

Fig. 5 defines an ontology describing the semantics of service advertisements and requests using the Ontology Web Language (*OWL*). This ontology is similar to the latest *OWL-S* 1.0 [17] (*OWL-S* is an *OWL*-based Web service ontology).

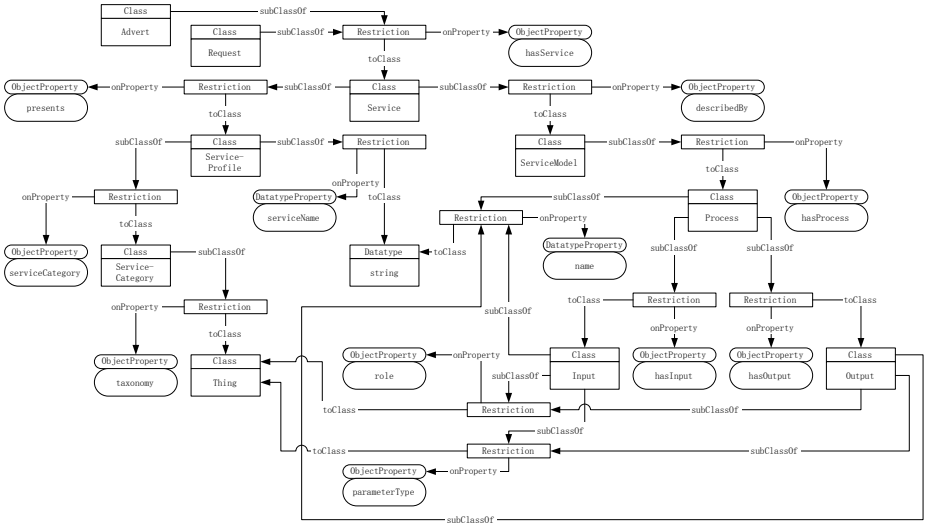

Fig. 5. The ontology for describing Web service advertisements and requests

The ontology presents a meta-model of Web service descriptions. Important concepts that participate in the satisfiability evaluation are specified using capacity interpretation or restriction interpretation. Each important concept is also assigned a specific satisfiability evaluation process. The specification of semantic interpretations and the assignment of satisfiability evaluation processes are described by means of *CRI* models.

Definition 9 (CRI model): *A Capacity-Restriction Interpretation (CRI) model, CRIM = (O, imap, emap), is a capability matching oriented structure, where:*

- *O is a Web service ontology (e.g., the one defined in Fig. 5) for describing the semantics of service advertisements and requests.*
- *imap: C* $\rightarrow$ *{capacity, restriction} is a partial function that relates a concept with a semantic interpretation (capacity interpretation or restriction interpretation). C is a concept defined in O.*

- *emap: C → {equivalence, plug-in, subsumption} is a partial function that relates a concept with a satisfiability evaluation process (equivalence evaluation, plug-in evaluation, or subsumption evaluation). C is a concept defined in O.*

Different satisfiability evaluation processes result in different impacts on capability matching. Subsumption evaluation requires that a concept from the request should subsume or equivalent to the corresponding concept from the advertisement (like the concept subsumption reasoning approach), while plug-in evaluation has a right opposite criterion (like the concept coverage comparing approach). Finally, equivalence evaluation asks for an equivalent relation between two involved concepts.

CRI models are used to describe Web service semantics, especially focusing on the specification of semantic interpretations and the assignment of satisfiability evaluation processes. Its significance lies in that it is the first time, to our best knowledge, to explicitly separate the assignment of satisfiability evaluation processes from the semantics representation of Web services, such that the satisfiability evaluation can be changed or improved independent of the service semantics representation. Many researches implicitly think that all concepts have capacity interpretations, and therefore, the satisfiability evaluation is fixed to the comparison of concept coverage. Some researches adopting *DL*s lack a flexible assignment mechanism for different satisfiability evaluation processes, only allowing concept subsumption reasoning. *CRI* models improve the matching accuracy by allowing dynamic specification of semantic interpretations and flexible assignment of satisfiability evaluation processes.

In a *CRI* model, the ontology, semantic interpretations and satisfiability evaluation processes are dynamically bound. The deciding of the most appropriate assignment of satisfiability evaluation processes mostly depends on the semantic interpretations of concepts involved, but the concept meanings and the related matching requirements also have their impacts. At present, our research suggests the following *CRI* model:

- *O*: The ontology defined in Fig. 5.
- *imap*: (1) *ServiceCategory*, *taxonomy*, ***Thing*** → capacity; (2) *Input*, *parameterType*, ***Thing*** → restriction; (3) *Output*, *parameterType*, ***Thing*** → restriction.
- *emap*: (1) *ServiceCategory*, *taxonomy*, ***Thing*** → plug-in; (2) *Input*, *role*, ***Thing*** → equivalence; (3) *Input*, *parameterType*, ***Thing*** → subsumption; (4) *Output*, *role*, ***Thing*** → equivalence; (5) *Output*, *parameterType*, ***Thing*** → subsumption.

Note that the bold literals are concepts under consideration. We use a prefix before each *Thing* concept to specify which *Thing* we are referring to (Fig. 5). Explanations for this suggested *CRI* model are given below.

The term *taxonomy* represents service category. A more general *taxonomy* means a more powerful service (e.g. *renting_vehicle* can satisfy *renting_car*). We specify *taxonomy* with a capacity interpretation. *parameterType* represents range restriction for an input/output parameter (e.g. we may need an *integer* input, or we plan to produce a *float* output). We specify both *parameterType*s to have restriction interpretations.

The assignment of satisfiability evaluation processes has close relations with semantic interpretations specified by function *imap*. Realizing that the *taxonomy* has a capacity interpretation, we examine Table 3 with both c_1 and c_2 subject to capacity interpretations. We find that c_1 is satisfied by c_2 only when they have equivalent or subsumed relations. As such, concept *taxonomy* matches successfully only when the *taxonomy* from a service request could be plugged in or equivalent to the correspond-

ing *taxonomy* from a service advertisement. Theorem 1 and Theorem 3 can infer this conclusion in a more convenient way. Therefore, we assign plug-in evaluation process to *taxonomy*. Similarly, we assign subsumption evaluation process to *parameterType*. This time we check Table 3 based on restriction interpretations of c_1 and c_2. Alternatively, we use Theorem 1 and Theorem 4 to infer it directly.

We have two more mappings about *role*. A *role* represents the business role of *Input*s or *Output*s of a Web service process. In the satisfiability evaluation, two *roles* from an advertisement and a request respectively should be similar or equal in meaning. We assign equivalence evaluation process to them.

3.4 Capability Matching Algorithm

The capability matching algorithm examines each advertisement in the Web service repository to find whether there is one fully or partially satisfying the given request. For each advertisement and request pair, the algorithm does the satisfiability evaluation based on the *ServiceProfile* and *ServiceModel* semantics (Fig. 5). For the *ServiceProfile* part, the algorithm performs plug-in evaluation for *taxonomy* as our *CRI* model suggests. But for the *ServiceModel* part, it is more complex.

A *ServiceModel* can have several *Process*es, and each *Process* may have multiple *Input*s/*Output*s. For full capability matching, the algorithm checks whether for each *Process* declared in the request, the advertisement can offer a satisfying *Process*. However, for partial capability matching, the criterion is reduced to whether there is at least one *Process* declared in the request such that the advertisement can offer a satisfying *Process*. Partial capability matching serves for service compositions.

No matter whether using full or partial capability matching, the algorithm checks all *Input*s and *Output*s in the *Process* pair being examined. A match is recognized when the following two conditions are met: (i) for each *Input* in the *Process* from the advertisement, there is a matched *Input* in the *Process* from the request (it guarantees that the request can provide sufficient *Input*s); (ii) for each *Output* in the *Process* from the request, there is a matched *Output* in the *Process* from the advertisement (it guarantees that the advertisement can provide sufficient *Output*s).

For the *Input*/*Output* part, the algorithm performs equivalence evaluation for *role* and subsumption evaluation for *parameterType*, following our *CRI* model. Due to space limitation, only the pseudo code for full capability matching algorithm is given below.

```
List capabilityMatching(req) {
  List result = new List();
  forall adv in advertisement_repository do {
    if checkProfile(getProfile(req), getProfile(adv)) == true
      && checkModel(getModel(req), getModel(adv)) == true
    then result.add(adv);
  }
  return result;
}
boolean checkProfile(reqProfile, advProfile) { ... }
boolean checkModel(reqModel, advModel) {
  forall reqProcess in getAllProcesses(reqModel) do {
    if not exists(advProcess) in getAllProcesses(advModel)
      such that checkProcess(reqProcess, advProcess) == true
```

```
      then return false;
    }
    return true;
  }
  boolean checkProcess(reqProcess, advProcess) {
    forall advInput in getAllInputs(advProcess) do {
      if not exists(reqInput) in getAllInputs(reqProcess
        such that checkInput(advInput, reqInput) == true
      then return false;
    }
    forall reqOutput in getAllOutputs(reqProcess) do { ... }
    return true;
  }
  boolean checkInput(advInput, reqInput) {
    if equivalent(getRole(advInput), getRole(reqInput)) == true
      && subsuming(getParamType(advInput),
      getParamType(reqInput)) == true
    then return true else return false;
  }
  boolean checkOutput(reqOutput, advOutput) { ... }
```

4 Implementation

Our Web service capability matching system adopts a three-tier architecture as illustrated in Fig. 6. *System Service Layer* provides four functionalities: advertisement browsing, advertisement/request consistency checking, request matching, and service composition. *Ontology Matching Layer* realizes ontology consistency checking and ontology structure matching for Web services based on *xlinkit* technology. *Satisfiability Evaluation Layer* is responsible for the satisfiability evaluation.

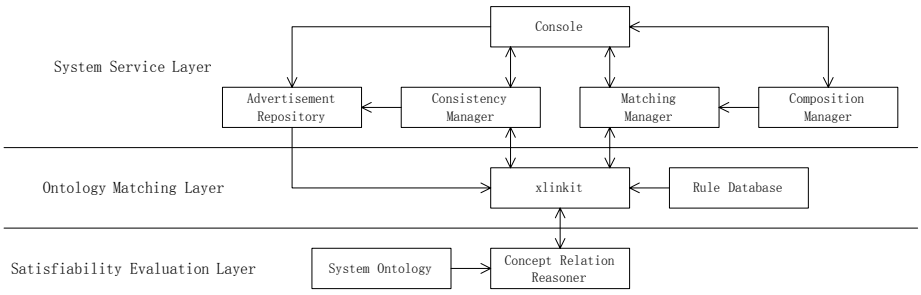

Fig. 6. The architecture of our Web service capability matching system

Detailed implementation considerations including the adaptation of *xlinkit* to service matching purposes are explained in [13]. The applications of *xlinkit* in the software engineering field can be found in [7][8][9].

5 Evaluation

Paolucci proposed four criteria in [11] for evaluating a service capability matching system: (i) the system should support semantics matching between advertisements

and requests based on ontology; (ii) the system should minimize false positives and false negatives; (iii) the system should encourage service providers and users to be honest with their descriptions at the cost of paying the price of either not being matched or being matched inappropriately; and (iv) the matching process should be efficient.

Our matching system strives to satisfy all the four criteria. Firstly, the matching is based on an *OWL* ontology. The system can perform semantic inferences leading to the recognition of capability matching despite of their syntactic differences. Secondly, the use of *OWL* also supports accuracy: no matching is recognized when the satisfiability criteria cannot be met for a given advertisement and request pair. Thirdly, dishonesty behaviors (e.g. arbitrarily aggrandizing capabilities) by service providers and users will lead to no matching or inappropriate matching, which will harm them eventually. Finally, in order to increase performance, our system adopts an efficient *xlinkit* tool for performing ontology structure matching between advertisements and requests, and uses a special concept relation reasoner for the kernel satisfiability evaluation.

In addition, our matching system has two more features: (i) the use of the notation of capacity interpretation and restriction interpretation to adapt to different contexts for improving the accuracy of semantics representation; and (ii) utilization of *CRI* models to support flexible assignment of satisfiability evaluation processes for improving the accuracy of satisfiability evaluation.

6 Conclusions and Future Work

In this paper, we studied several Web service capability matching models, and classified their satisfiability evaluation techniques into three approaches. We analyzed their limitations under two semantic interpretations: restriction interpretation and capacity interpretation. To address these limitations, we proposed a unified Web service capability matching model, in which the satisfiability evaluation can be performed in a uniform way based on semantic interpretations of concepts. On the basis of the satisfiability result table and satisfiability theorems, we have proposed *CRI* models as a promising solution to the Web service capability matching problem. The corresponding algorithm can be implemented based on *xlinkit* technology.

At present, our implementation is subject to a communication bottleneck between the *Ontology Matching Layer* and the *Satisfiability Evaluation Layer*. We are working at various means to alleviate this problem such that our prototype could be refined into a practical semantic Web service tool. More features (e.g., satisfiability evaluation between capacity interpretation and restriction interpretation) will be incorporated in the refined *CRI* model. We also plan to extend the proposed semantic evaluation to the support of mobile and pervasive services [3][14].

Acknowledgements

This work was supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China, under Project HKUST6170/03E. The authors would also like to thank Min Ye (csmarco@cs.ust.hk) for his assistance with the system implementation.

References

1. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P. Patel-Schneider, *The Description Logic Handbook*, Cambridge University Press, 2002.
2. Xiang Gao, Jian Yang, Mike P. Papazoglou, The Capability Matching of Web Services, *Proceedings of the 4th International Symposium on Multimedia Software Engineering* (*MSE'02*), Newport Beach, California, USA, Dec 2002.
3. Dickson K.W. Chiu, S.C. Cheung, Eleanna Kafeza and H.F. Leung, A Three-tier View-based Methodology for M-Services Adaptation, *IEEE Transactions on Systems, Man, and Cybernetics (Part A)*, 33 (6), November 2003, pp. 725-741.
4. Lei Li, Ian Horrocks, A Software Framework for Matchmaking Based on Semantic Web Technology, *Proceedings of the WWW 2003 Conference*, Budapest, May 2003, pp 331-339.
5. S.A. Mcllraith, T.C. Son, H. Zeng, Semantic Web Services, *IEEE Intell Sys 16*(*2*): 46-53.
6. Brahim Medjahed, et al, Composing Web Services on the Semantic Web, *VLDB Journal* (*2003*) (*Special Issue on the Semantic Web*) *12*: 333-351.
7. C. Nentwich, W. Emmerich, A. Finkelstein, Consistency Management with Repair Actions, *Proceedings of the 25th International Conference on Software Engineering* (*ICSE'03*), Portland, Oregon, USA, May 2003.
8. C. Nentwich, et al, xlinkit: A Consistency Checking and Smart Link Generation Service, *ACM Transactions on Internet Technology 2*(*2*): 151-185, May 2002.
9. C. Nentwich, W. Emmerich, A. Finkelstein, Static Consistency Checking for Distributed Specifications, *Proceedings of the 16th International Conference on Automated Software Engineering* (*ASE*), pp 115-124, Coronado Island, CA, Nov 2001.
10. T.R. Payne, et al, Advertising and Matching DAML-S Service Descriptions, *Proceedings of the International Semantic Web Working Symposium*, Stanford, CA, July, 2001.
11. Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, Katia Sycara, Semantic Matching of Web Services Capabilities, *Proceedings of the 1st International Semantic Web Conference*, Sardinia, Italy, June 2002, pp 318-332.
12. Katia Sycara, Mattheus Klusch, Seth Widoff, Janguo Lu, Dynamic Service Matchmaking among Agents in Open Information Environments, *ACM SIGMOD Record* (*Special Issue on Semantic Interoperability in Global Information System*) *28*(*1*): 47-53, 1999.
13. Chang Xu, S.C. Cheung, Xiangye Xiao, Capability Matching of Web Services, *Technical Report HKUST-CS04-08*, Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong, July 2004.
14. Chang Xu, S.C. Cheung, Cindy Lo, K.C. Leung, Jun Wei, Cabot: On the Ontology for the Middleware Support of Context-Aware Pervasive Applications, *Proceedings of the Workshop of Building Intelligent Sensor Networks (BISON'04)*, October 2004, Wuhan, China.
15. DAML+OIL Language, http://www.daml.org/2001/03/daml+oil-index.html, 2001.
16. DAML-S: Semantic Markup for Web Services, http://www.daml.org/services/daml-s/0.9/daml-s.html, May 2003.
17. OWL-S: Semantic Markup for Web Services, http://www.daml.org/services/owl-s/1.0/owl-s.html, Nov 2003.
18. W3C (2003) Simple Object Access Protocol (SOAP), http://www.w3.org/TR/soap.
19. W3C Web Service Description Language (WSDL), http://www.w3.org/TR/wsdl.
20. W3C Universal Description, Discovery, and Integration (UDDI), http://www.uddi.org.