Chapter 10

Service Discovery

Luis Botelho, Alberto Fernández, Benedikt Fries, Matthias Klusch, Lino Pereira, Tiago Santos, Pedro Pais, Matteo Vasirani

10.1 Introduction

Semantic service discovery is the process of locating Web Services based on the description of their functional and non-functional semantics. Both service oriented computing and the semantic Web envision intelligent agents to proactively pursue this task on behalf of their clients. Service discovery can be performed in different ways depending on the service description framework, on means of service selection, and on its coordination through assisted mediation or in a peer-to-peer fashion.

In the CASCOM system, semantic service discovery is realised by the interplay between a service discovery agent (SDA), a project distributed service repository (WSDir), and a semantic service matchmaker (SMA). On request, the SDA searches for relevant services in both the WSDir and in its own local service repository. Service selection is implemented through means of a rather coarse-grained keyword-based matching of services as a quick filtering operation by the SDA which is complemented by a more fine-grained logic-based analysis of service semantics by the SMA.

This chapter is structured as follows. First, we provide an overview of the discovery approach based on the interaction between SDA, WSDir, and SMA. This is followed by a more detailed description of both agents, the SDA and the SMA with focus on the integrated service matchmaking algorithms. The chapter ends with a conclusions section.

10.2 Overview

In the CASCOM system, service discovery results of the cooperation between the requester (Personal Agent), the Service Discovery Agent (SDA) that coordinates

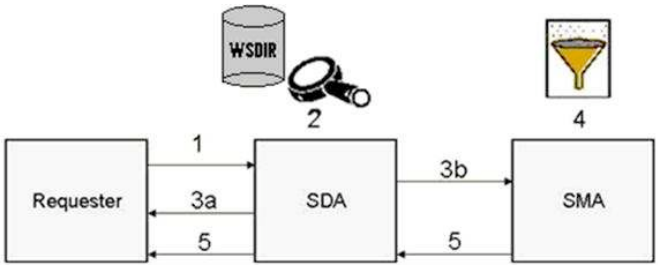

Figure 10.1: Service selection process in CASCOM

the search process, the WSDIR (a distributed service directory described in Chapter 9), and the Service Matchmaking Agent (SMA) with several integrated matchmaking algorithms. The WSDIR stores advertised OWL-S service descriptions and supports searching for them according to both functional and non-functional service semantics. The typical semantic service discovery process in CASCOM is depicted in Figure 10.1.

In step 1, the requester asks the SDA for services or service providers matching specified criteria. In step 2, the SDA extracts the specified service category from the received request and consults its own service database and the WSDIR to acquire the services that match the specified category. This coarse-grained step corresponds to a quick filtering operation that is part of the whole service selection process.

Next, the SDA matches the services that passed the quick filter with the remaining criteria specified in the request. Service selection can be entirely performed by the SDA itself based on its internal service repository or consultance of the distributed service directory (WSDir), or by calling the semantic service matchmaker SMA depending on the service selection criteria specified in the received request.

If all specified criteria involve only simple matching operations (e.g., category matching) the matching process is done by the SDA only. If the specified criteria require complex matching processes, such as preconditions and effects matchmaking, subsumption reasoning, or role-based matchmaking, the matching process is performed by the SMA. If the SDA can solely perform the matching process without the SMA, it sends the resultant set of services to the requester (step 3a). If it is necessary to involve the SMA, the SDA sends the discovered set of services together with the original request to the SMA (step 3b).

In the fourth step, the SMA selects the set of services that match according to the specified criteria. In step 5, the results are returned to the SDA and consequently to the requester. Although context processing is not explicitly represented in Figure 10.1, each agent acquires relevant context information, possibly from the context acquisition and management system (see Chapter 13), and uses it to better adapt its performance to the current situation.

In summary, the SDA can initially prune the set of available services to those that match according to a given criteria. The SDA may trigger an additional logic-based matching process on those initially selected services by the SMA. Finally, the SDA internal service database and the WSDIR store service descriptions according to different policies and ensure privacy and security.

Finally, please note that the separation between the SDA, the SMA and the WSDir, although allowing for better and easier system development and management, also has the disadvantage of requiring more intense communication of service descriptions. Besides, it has to be appropriately upgraded to also work for service discovery in totally distributed networks without service directories.

10.3 The CASCOM Service Discovery Agent

Besides service discovery, the SDA may also be used by service providers to register their services in the WSDIR or in its database. This is an important feature because the WSDIR, being a Web Service, does not provide an agent interface. Therefore, if a service registration agent interface with WSDir is desired, the SDA may be used. Currently, the SDA offers the following functionalities:

- Request a set of complete service descriptions, service profiles, service processes, or service groundings that match the specified criteria
- Register a complete service description, a service profile, a service process or service grounding.
- Associate a profile, a process, or grounding to a specified service
- Remove a complete service description, or a service profile, service process or service grounding
- Register or remove a service provider.

The separated manipulation of the several elements of the service description (service profile, service process and service grounding) is an important feature because a service can have several profiles or several groundings. Unfortunately, this separation is possible only when using the SDA internal service database; The WSDir does not support such separation. The SDA, as all other agents of the CASCOM service coordination system, uses the FIPA ACL communication language with FIPA SL contents. Figure 10.2 shows an example of a message used to request a set of complete service descriptions that match a given service profile, which is specified through an URL.

The interaction with the Service Discovery Agent is based on the FIPA Request interaction protocol, which is used for an agent to request another one to perform some specified action Figure 10.3.

The receiving agent can accept or refuse to perform the requested action. If the receiver refuses to execute the requested action, the conversation stops,

```
(REQUEST
 :sender (agent-identifier :name client@cascom)
 :receiver (set (agent-identifier :name sda@cascom))
 :content "((action
     (agent-identifier :name sda@cascom)
     (getMatchingServices
       :profileURI
         \"http://www.daml.org/services/owl-s/1.1/BravoAirProfile.owl\"
       :profileLocation \"http//localhost/BravoAirProfile.owl
       :useMatchmaker false\")))"
 :language fipa-sl
 :protocol fipa-request)
```

Figure 10.2: Example request message for relevant services

otherwise, the receiver must try to execute it. In case of success, the receiver sends the results to the sender or merely informs it that the action was successfully executed. In case of failure, the receiver must inform the sender that the execution failed, and the reasons of failure.

The SDA was completely implemented in JAVA. It uses the OWL-S API[1] to read and process OWL-S service descriptions[2]. For results of evaluating the SDA discovery agent, we refer to Chapter 16.

10.4 The CASCOM Service Matchmaker

Within the CASCOM coordination system, the semantic service selection functionality is provided by the Service Matchmaking Agent (SMA). This agent is constituted by three building blocks, each of them corresponding to a different Semantic Web Service matchmaker: the hybrid service I/O matchmaker OWLS-MX, the Precondition and Effect matchmaker PCEM, and the Role-based matchmaker ROWLS.

The OWLS-MX matchmaker (see Section 10.5) performs hybrid service signature matching through complementing logic-based semantic I/O matching with syntactic token-based similarity metrics to obtain the best of both worlds - description logics and information retrieval. The Precondition and Effect matchmaker PCEM (see Section 10.6) exploits pre-conditions and effects of service descriptions, converting them in logic predicates and using a Prolog reasoner to determine exact and inferred relations. The Role-based matchmaker ROWLS (see Section 10.7) exploits the structuring of services in terms of organisational concepts such as roles

[1] See http://www.mindswap.org/

[2] Further details about SDA (including a Demo) can be found at http://www.we-b-mind.org/sda

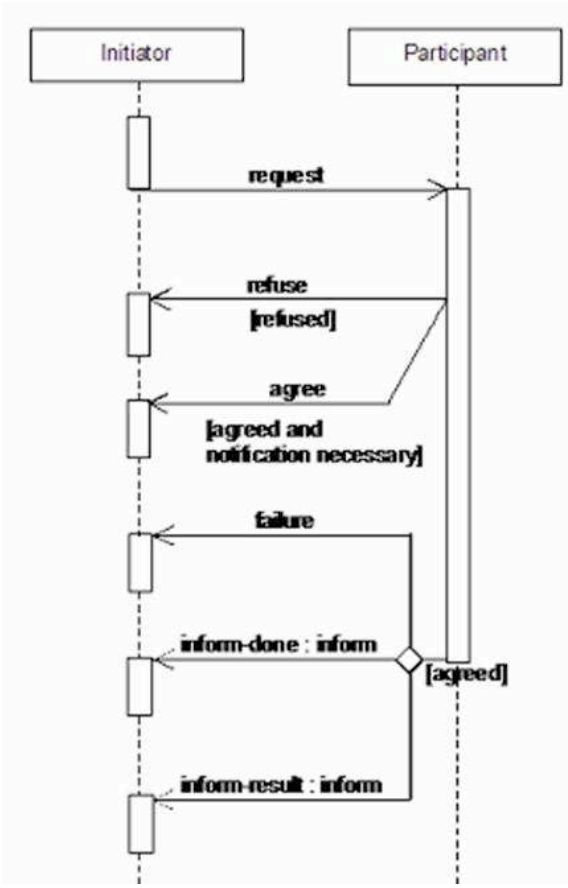

Figure 10.3: FIPA-Request Protocol of SDA

and types of social interactions.

10.4.1 Configurations

How to properly integrate these different matchmakers into the CASCOM service matchmaker? In the project, we implemented the following configurations:

1. *Sequential*. Using the three matchmakers sequentially, where each matchmaker acts as a pre-filter of the next one in the sequence. A possible configuration can be the one depicted in Figure 10.4, where the role-based matchmaker possibly reduces the number of input services of OWLS-MX, which in turn reduces the number of input services of the PCEM matchmaker. This order has been chosen on the basis of the computational complexity of the

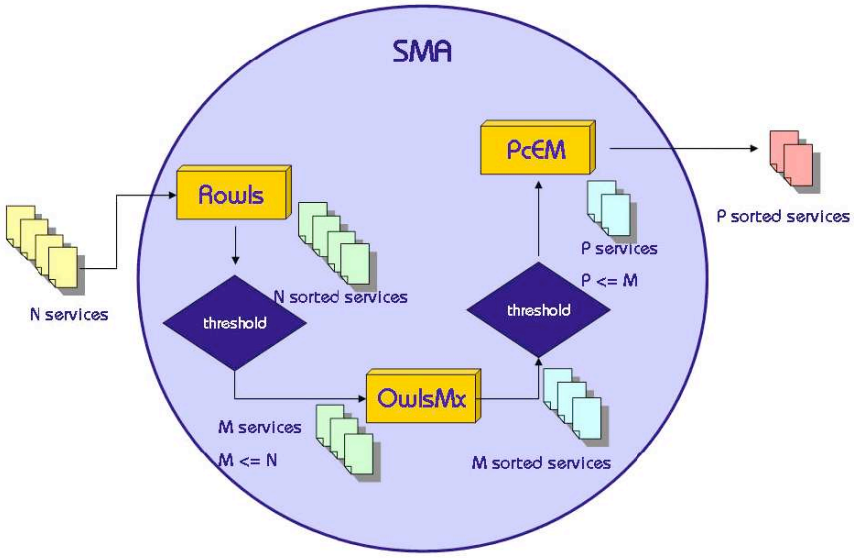

Figure 10.4: Sequential configuration

matchmakers.

2. *Concurrent.* Running all three matchmakers with the same set of services and then combining the returned degrees of match with an aggregation function (Figure 10.5).

At the time the SMA is called by another agent, the individual configuration of utilizing the matchmaking modules can be chosen by setting a special parameter in the request message.

Sequential Matching by the SMA A possible configuration is executing firstly the role-based matchmaker. The role-based matchmaker assigns to every service a real valued number between 0 and 1.

Then, a reduced set of these services are given to OWLS-MX as input. This reduced set can be composed by

- All the services which have a role-based degree of match greater than a threshold
- The first K services of the ordered set, where K is fixed
- The first M services of the ordered set, where M is a percentage of the original set size

OWLS-MX performs the matchmaking with this reduced set of services, and returns an ordered list of services, depending of the parametrization of the matchmaker. There are five possible instantiations of OWLS-MX, M0 to M4, where M0 stands for a purely logic-based semantic matching, while the others instantiations additionally use a token-based syntactic similarity metric (namely, the known "Loss-of-information", "Extended Jacquard", "Cosine", and "Jensen-Shannon divergence" based similarity measure). The configurable parameters comprises the minimum degree of match, the matchmaker type and, in case of use of syntactic similarity metric, the syntactic similarity threshold. The minimum degree of match can be *EXACT*, *PLUG_IN*, *SUBSUMES*, *SUBSUMED_BY* or *NEAREST_NEIGHBOUR*, while the syntactic similarity threshold is a real value used only for *SUBSUMED_BY* or *NEAREST_NEIGHBOUR* match.

Finally, the output of OWLS-MX is passed to the PCEM matchmaker as input. Since this matchmaker does not return a real valued degree of matching, the SMA calculates the (final) matching value as follows (cf. Section 10.6)

$$(\textit{Precondition Exact Matching} \vee \textit{Precondition Reasoning Matching}) \wedge (\textit{Effect Exact Matching} \vee \textit{Effect Reasoning Matching})$$

The return value of this formula (TRUE or FALSE) is used to create the final returned set of services. We have chosen this order of the matchmakers because the computational complexity of the role-based matchmaker ROWLS is lower than the computational complexity of OWLS-MX, which is in turn lower than the computational complexity of the PCEM matchmaker. Hence, the matchmaker that is supposed to work with the greatest set of services is the role-based one (ROWLS); OWLS-MX will work with a smaller set of services, and PCEM with an even smaller one.

The parameters that need to be set in this configuration of the SMA are

- The filter parameter of the role-based matchmaker ROWLS (threshold or K or M);
- The type of OWLS-MX matchmaker (M0, M1, M2, M3, M4), the minimum degree of match and the similarity threshold.

Aggregated Matching by the SMA Another possible configuration of the SMA is the execution of all three matchmakers in parallel with the same set of services, and the aggregation of the results by means of special aggregation function.

For this purpose, it is necessary that every matchmaker assigns a matching value for every service. However, only the role-based matchmaker returns a real number between 0 and 1 as degree of match, while a degree of match for the PCEM matchmaker can be generated applying the above logic formula (returning a value equal to either 0 or 1). The OWLS-MX, on the other hand, returns the degree of match as a category, which can be *EXACT*, *PLUG_IN*, *SUBSUMES*, *SUBSUMED_BY* and *NEAREST_NEIGHBOUR*, while all the other services that are not returned can be considered having a degree of match equal to *FAIL*. So,

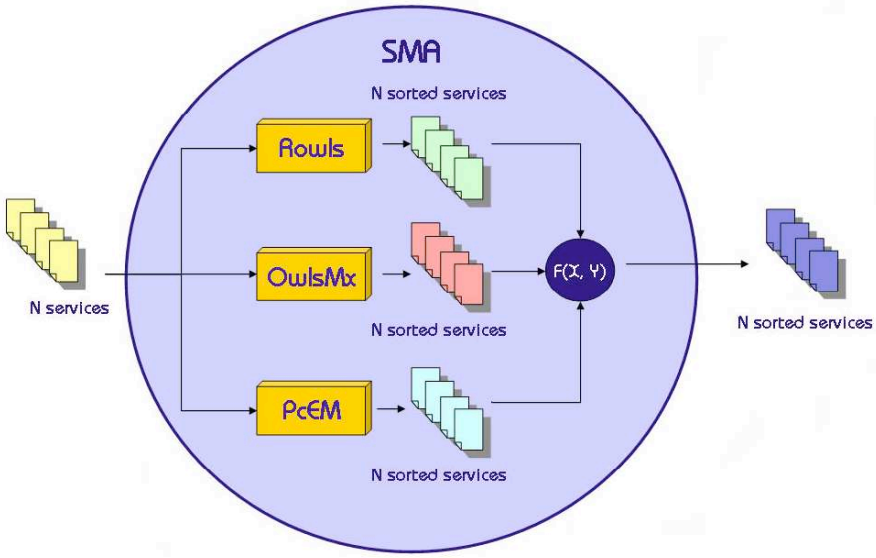

Figure 10.5: Aggregation of the three matchmakers' results by the SMA

for the OWLS-MX it is necessary to assign to every category, a real valued number between 0 and 1 (for example, $EXACT = 1$, $PLUG\_IN = 0.7$, $SUBSUMES = 0.5$, $SUBSUMED\_BY = 0.3$, $NEAREST\_NEIGHBOUR = 0.1$, $FAIL = 0$).

Having assigned a degree of match for every service matched by every matchmaker, it is possible to combine the three results with an aggregation function, and order the original set of services on the basis of the aggregation value. Possible aggregation functions are the minimum, the product, the weighted product or more complex ones.

The parameters that need to be set in this configuration are

- The type of OWLS-MX matchmaker (M0, M1, M2, M3, M4), the minimum degree of match and the similarity threshold
- The aggregation function

Predefined Configurations In order to make easier to select the parameters of the different components of the SMA, several configurations have been programmed, which can be selected whenever the *getMatchingServices* action is requested (through the *FIPA-Request* protocol). Figure 10.6 shows the predefined configurations.

For each possible configuration, the values for the different parameters of its internal components are reported. In case that no parameter is provided in the invocation of the matchmaking, a default configuration is selected.

SMA Parameter	MODE {sequential| aggregation}	ROWLS Threshold 0..1	OR	k-filter Integer	OR	%-filter 0..100	OWLS-MX Type {M0,..M4}	Min DOM {Exact|plugin|subsumes|subsumed-by| nearest-neighbour}	Simil. Threshold 0..1	Aggregation function {min|prod|Lukasiewicz| weighted average}
DEFAULT	Sequential	---		---		60	M0	Exact	---	---
Exact	Sequential	1		---		---	M0	Exact	---	---
plugin	Sequential	0.7		---		---	M1	plugin	---	---
subsumes	Sequential	0.5		---		---	M2	subsumes	---	---
subsumed-by	Sequential	0.3		---		---	M3	subsumed-by	0.5	---
nearest-neighbour_1	Sequential	---		20		---	M4	nearest-neighbour	0.2	---
nearest-neighbour_2	Sequential	0.3		---		---	M2	nearest-neighbour	0.5	---
aggregation 1	Aggregation	---		---		---	M1	nearest-neighbour	0.7	Lukasiewicz Max(0,x+y-1)
aggregation 2	Aggregation	---		---		---	M2	nearest-neighbour	0.5	prod
aggregation 3	Aggregation	---		---		---	M3	nearest-neighbour	0.3	min
aggregation 4	Aggregation	---		---		---	M4	nearest-neighbour	0.5	0.5*OWLS-MX + + 0.3*ROWLS + 0.2*PcEM
owls-mx	---	---		---		---	M4	nearest-neighbour	0.5	---
rowls+owlsmx	Sequential	0.4		---		---	M4	nearest-neighbour	0.5	---
owls-mx+pcem	Sequential	---		---		---	M4	nearest-neighbour	0.5	---

Figure 10.6: SMA predefined configurations

10.4.2 SMA Interface

The communication protocol followed by the SMA is the FIPA-Request protocol. Only one action can be requested to the SMA, `getMatchingServices`. It accepts as parameter a URI of a OWL-S service profile as query (request), one or more service profiles (services) and, optionally, a parameter that defines the configuration of the three matchmakers (matchType). Figure 10.7 shows an example of a message sent to the SMA.

After having received the message, the SMA performs the match of the request against the list of service descriptions and, in case of successful execution of the matchmaking operation, it returns a sorted set of services (Figure 10.8).

10.5 Hybrid Semantic Service Matchmaker OWLS-MX

One option of the CASCOM matchmaker agent SMA to find relevant OWL-S services in the semantic Web is through its OWLS-MX matchmaker module. This module exploits both logic-based reasoning and content-based information retrieval (IR) techniques for OWL-S service profile I/O matching. In the following, we define the hybrid semantic filters of OWLS-MX, the generic matching algorithm, and its five variants according to the used IR similarity metrics. Familiarity with OWL-S and description logics is assumed. More details and evaluation results can be found in [4].

10.5.1 Hybrid Matching Filters

OWLS-MX computes the degree of semantic matching for a given pair of service advertisement and request by successively applying five different filters EXACT, PLUG IN, SUBSUMES, SUBSUMED-BY and NEAREST-NEIGHBOR. The first three are logic-based only whereas the last two are hybrid due to the required additional computation of syntactic similarity values.

```
(REQUEST
  :sender(agent-identifier :name sda@host1:1099/JADE
                :addresses (sequence http://host1:7778/acc))
  :receiver(set(agent-identifier :name sma@host2:1099/JADE))
  :content "(
    (action(agent-identifier :name sda@host1:1099/JADE
                  :addresses (sequence http://host2:7778/acc))
      (getMatchingServices
        :request "http://query.owl"
        :services (sequence "http://service1.owl"
             "http://service2.owl"..."http://serviceN.owl")
        :matchType DEFAULT)))"
  :language fipa-sl
  :ontology cascom-ontology
)
```

Figure 10.7: Request message to SMA

Let T be the terminology of the OWLS-MX matchmaker ontology specified in OWL-DL (SHOIN(D)); CT_T the concept subsumption hierarchy of T; $LSC(C)$ the set of least specific concepts (direct children) C' of C, i.e. C' is immediate sub-concept of C in CT_T; $LGC(C)$ the set of least generic concepts (direct parents) C' of C, i.e., C' is immediate super-concept of C in CT_T; $Sim_{IR}(A, B) \in [0, 1]$ the numeric degree of syntactic similarity between strings A and B according to chosen IR metric IR with used term weighting scheme and document collection, and $\alpha \in [0, 1]$ given syntactic similarity threshold; $\doteq$ and $\succeq$ denote terminological concept equivalence and subsumption, respectively.

Exact Match. Service S EXACTLY matches request R $\Leftrightarrow \forall\ \text{IN}_S\ \exists\ \text{IN}_R$: $\text{IN}_S \doteq \text{IN}_R \wedge \forall\ \text{OUT}_R\ \exists\ \text{OUT}_S$: $\text{OUT}_R \doteq \text{OUT}_S$. The service I/O signature perfectly matches with the request with respect to logic-based equivalence of their formal semantics.

Plug-in Match. Service S PLUGS INTO request R $\Leftrightarrow \forall\ \text{IN}_S\ \exists\ \text{IN}_R$: $\text{IN}_S \succeq \text{IN}_R \wedge \forall\ \text{OUT}_R\ \exists\ \text{OUT}_S$: $\text{OUT}_S \in \text{LSC}(\text{OUT}_R)$. Relaxing the exact matching constraint, service S may require less input than it has been specified in the request R. This guarantees at a minimum that S will be executable with the provided input iff the involved OWL input concepts can be equivalently mapped to WSDL input messages and corresponding service signature data types. We assume this as a necessary constraint of each of the subsequent filters.

In addition, S is expected to return more specific output data whose logically defined semantics is exactly the same or very close to what has been requested by the user. This kind of match is borrowed from the software engineering domain, where software components are considered to plug-in match

```
(INFORM
  :sender(agent-identifier :name sma@host2:1099/JADE
                :addresses (sequence http://host2:7778/acc))
  :receiver(set(agent-identifier :name sda@host1:1099/JADE))
  :content "(
    (result
      (action(....))
      (sequence "http://service2.owl"
           "http://serviceN.owl"..."http://service1.owl")))"
  :language fipa-sl
  :ontology cascom-ontology
)
```

Figure 10.8: Inform message from SMA

with each other as defined above but not restricting the output concepts to be direct children of those of the query.

Subsumes Match. Request R SUBSUMES service S $\Leftrightarrow \forall$ IN$_S$ $\exists$ IN$_R$: IN$_S$ $\succeq$ IN$_R$ $\wedge \forall$ OUT$_R$ $\exists$ OUT$_S$: OUT$_R$ $\succeq$ OUT$_S$. This filter is weaker than the plug-in filter with respect to the extent the returned output is more specific than requested by the user, since it relaxes the constraint of immediate output concept subsumption. As a consequence, the returned set of relevant services is extended in principle.

Subsumed-by Match. Request R is SUBSUMED BY service S $\Leftrightarrow \forall$ IN$_S$ $\exists$ IN$_R$: IN$_S$ $\succeq$ IN$_R$ $\wedge \forall$ OUT$_R$ $\exists$ OUT$_S$: (OUT$_S$ $\doteq$ OUT$_R$ $\vee$ OUT$_S$ $\in$ LGC(OUT$_R$)) $\wedge$ SIM$_{IR}$(S, R) $\geq \alpha$. This filter selects services whose output data is more general than requested, hence, in this sense, subsumes the request. We focus on direct parent output concepts to avoid selecting services returning data which we think may be too general. Of course, it depends on the individual perspective taken by the user, the application domain, and the granularity of the underlying ontology at hand, whether a relaxation of this constraint is appropriate, or not.

Logic-Based Fail. Service S fails to match with request R according to the above logic-based semantic filter criteria.

Nearest-Neighbor Match. Service S is NEAREST NEIGHBOR of request R $\Leftrightarrow \forall$ IN$_S$ $\exists$ IN$_R$: IN$_S$ $\succeq$ IN$_R$ $\wedge \forall$ OUT$_R$ $\exists$ OUT$_S$: OUT$_R$ $\succeq$ OUT$_S$ $\vee$ SIM$_{IR}$(S, R) $\geq \alpha$.

Fail. Service S does not match with request R according to any of the above filters.

The OWLS-MX matching filters are sorted according to the size of results they would return, in other words according to how relaxed the semantic matching. In this respect, we assume that service output data that are more general than

requested relaxes a semantic match with a given query. As a consequence, we obtain the following total order of matching filters

$$\text{Exact} < \text{Plug-In} < \text{Subsumes} < \text{Subsumed-By} < \text{Logic-based Fail} < \text{Nearest-neighbor} < \text{Fail}.$$

10.5.2 OWLS-MX Matching Algorithm

The core idea of the OWLS-MX matchmaker is to complement crisp logic-based with approximate IR-based matching where appropriate to improve the retrieval performance. It takes any OWL-S service as a query, and returns an ordered set of relevant services that semantically match the query each of which annotated with its individual degree of logical matching, and the syntactic similarity value. The user can specify the desired degree, and individual syntactic similarity threshold.

For each given service query, OWLS-MX first classifies the respective service I/O concepts into its local matchmaker ontology. For this purpose, it is assumed that the type of computed terminological subsumption relation determines the degree of semantic relation between pairs of input and concepts.

Auxiliary information on whether an individual concept is used as an input or output concept by a registered service is attached to this concept in the ontology. The respective lists of service identifiers are used by the matchmaker to compute the set of relevant services that match the given query according to the five hybrid filters.

In particular, OWLS-MX does not only pairwisely determine the degree of logical match but syntactic similarity between the conjunctive I/O concept expressions in OWL-Lite. These expressions are built by recursively unfolding each query and service input (output) concept in the local matchmaker ontology. As a result, the unfolded concept expressions are including primitive components of a basic shared vocabulary only.

Any failure of logical concept subsumption produced by the integrated description logic reasoner of OWLS-MX will be tolerated, if and only if the degree of syntactic similarity between the respective unfolded service and request concept expressions exceeds a given similarity threshold.

10.5.3 OWLS-MX Variants

We implemented different variants of the generic OWLS-MX algorithm, called OWLS-M1 to OWLS-M4, each of which uses the same logic-based semantic filters but different IR similarity metric $\text{SIM}_{IR}(R, S)$ for content-based service I/O matching. Based on the experimental results of measuring the performance of similarity metrics for text information retrieval provided by Cohen et.al (2003), we selected the top performing ones to build these variants. The variant OWLS-MO performs logic-based only semantic service I/O matching.

OWLS-M0. The logic-based semantic filters EXACT, PLUG-IN, and SUBSUMES are applied as defined above, whereas the hybrid filter SUBSUMED-BY is utilized without checking the syntactic similarity constraint.

OWLS-M1 to OWLS-M4. The hybrid semantic matchmaker variants OWLS-M1, OWLS-M3, and OWLS-M4 compute the syntactic similarity value SIM_{IR} (OUT_S, OUT_R) by use of the loss-of-information measure, extended Jacquard similarity coefficient, the cosine similarity value, and the Jensen-Shannon information divergence based similarity value, respectively.

10.5.4 Implementation

We implemented the OWLS-MX matchmaker version 1.1 in Java using the OWL-S API 1.1 beta with the tableaux OWL-DL reasoner Pellet developed at the university of Maryland[3]. As the OWL-S API is tightly coupled with the Jena Semantic Web Framework, developed by the HP Labs Semantic Web research group[4], the latter is also used to modify the OWLS-MX matchmaker ontology. The OWLS-MX matchmaker is available as open source from the portal semwebcentral.org[5].

The results of the evaluation of OWLS-MX are provided in Chapter 16.

10.6 Service Precondition and Effect Matchmaker PCEM

Another option of the CASCOM matchmaker agent to determine the degree to which two service descriptions semantically match is to logically compare their pre-conditions and effects by means of its Pre-conditions and Effects Matchmaker (PCEM) module.

10.6.1 Motivation

As mentioned above, main goal of service matchmaking algorithms is to determine the degree to which two service descriptions match. In general, matchmaking algorithms receive a description that represents the requested service, and a set of published service descriptions; and returns the degree to which each of the published service descriptions matches the request. Service matchmaking is essential for service coordination because it helps select services that better satisfy the specified requirements.

Currently, most of the matchmaking algorithms, like LARKS (Language for Advertisement and Request for Knowledge Sharing) [13], the OWL-S/UDDI [8], the RACER [6], the MaMaS (MatchMaker-Service) [7], the HotBlu [2] and the OWLS-MX (Hybrid OWL-S Web Service Matchmaker) [4] take into account the

[3] cf. `http://www.mindswap.org`
[4] cf. `http://jena.sourceforge.net/`
[5] `http://projects.semwebcentral.org/projects/owls-mx/`

input parameters, the output parameters and the categories of the service descriptions being compared.

However, considering only their inputs, outputs and categories is often not enough for flexible service matchmaking. Sometimes, it is better to consider other service characteristics, such as service preconditions and service effects. Service matchmaking using service preconditions and effects may bring about three main advantages. First, the matching process may be much more precise than using inputs, outputs and categories alone. Matchmaking using inputs and outputs pay attention only to the classes of the service inputs and outputs and, at most, to constraints relating these parameters (see, for instance [13]). Since it is perfectly possible to have two services with input and output parameters of the same class and satisfying the same constraints, that play completely different roles, this simple matching process is not an accurate one.

The same kind of argument applies to the service categories. In most cases, service selection aims at identifying services that achieve a given effect if they are executed in some specified circumstances. This is exactly the information provided by service preconditions and effects. Second, it is not always necessary to know all the service input and output parameters. Not all services that achieve some desired effect have the same set of input and output parameters. For example, a given book selling service may require as input parameters the book name and the author name, while another one may also require the client's identification. If the request specifies only the book name and the author name, the two mentioned services would yield different matching degrees with the request. However, considering the user's actual needs, maybe each of them is as good as the other.

Third, using preconditions and effects allows the matching algorithm to reason about the compared preconditions and the compared effects. Usually, it is not necessary to find a service that achieves exactly the specified effect. Most often, like in plug-in IOPE matching (IOPE - Inputs, Outputs, Preconditions and Effects), it is enough to find a service whose effects imply the specified effects. By the same token, it is often enough to find a service whose preconditions are implied by the specified preconditions.

Only (additional) preconditions and effects matchmaking may take the mentioned facts into account. For example, if the client wants to find a service that can radiograph his finger, it is perfectly acceptable to select a service that can radiograph the client's hand or even the client's limbs. Since finger is neither a subclass of hand nor a subclass of limb, inputs, outputs and categories matchmaking algorithms would not select the service that radiographs hands or the service that radiograph limbs.

Given the described motivation, the CASCOM project decided to include preconditions and effects matchmaking in its service matchmaking agent SMA. This section describes the PCEM (Pre Conditions and Effects Matchmaking) component that implements this type of matchmaking process.

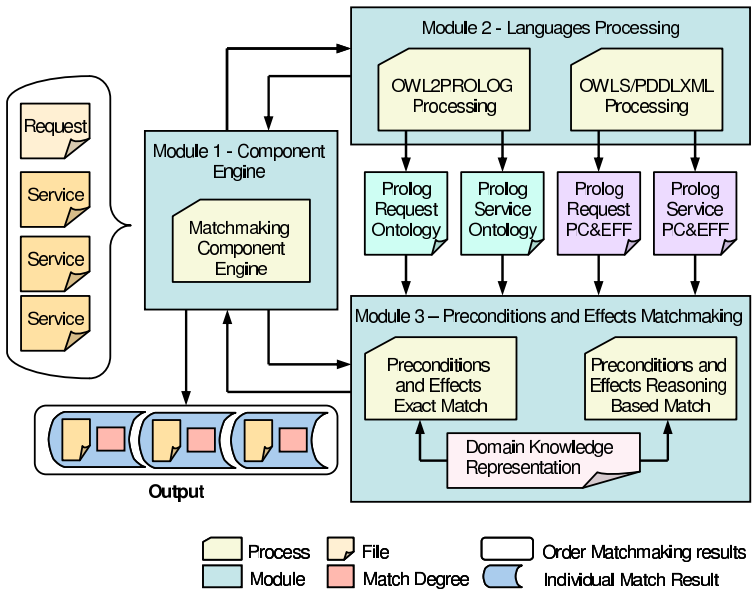

Figure 10.9: PCEM architecture

10.6.2 PCEM Architecture

This section describes the architecture of the developed preconditions and effects matchmaking component PCEM as shown in Figure 10.9.

The architecture is composed of three main modules. The first module ("Module 1 — Component Engine") controls the two other modules and determines the global matching degree of the service request with each of the available service descriptions. A detailed description of this module can be found in the section "Component Engine Module".

The second module ("Module 2 — Languages Processing") is responsible for language processing. It converts OWL ontologies into Prolog ontologies and OWL-S preconditions and effects into Prolog preconditions and effects. A detailed description of this module is presented in section "Languages Processing Module".

The third module ("Module 3 — Preconditions and Effects Matchmaking") implements the actual preconditions and effects matchmaking algorithms. It performs both exact matchmaking ("Pre-conditions and Effects Exact Match") and matchmaking using reasoning ("Pre-conditions and Effects Reasoning-based Match"). The latter uses both general purpose inference rules (e.g., deduction) and domain specific inference rules valid for a certain service effect or for certain precondition. These matchmaking algorithms are described in the section "Preconditions and Effects Matching".

10.6.3 PCEM Engine Module

The PCEM Engine module is responsible for controlling the other two modules and for determining the final matching degree of the matched service descriptions. All matchmaking requests sent to the component are received by the Component Engine. After receiving a matchmaking request the module retrieves the OWL-S description that represents the desired service specification and the set of OWL-S descriptions of the published available services. The descriptions in the received request are sent to the Languages Processing module where the Prolog language descriptions are generated.

The next step consist of sending a request to the Preconditions and Effects Matchmaking module for determining four partial matching degrees for each of the received service descriptions: exact preconditions matching degree, exact effects matching degree, preconditions reasoning matching degree and effects reasoning matching degree. After receiving the four partial matching degrees for all service descriptions, the Component Engine computes the global matching degree for each received service description and returns the list containing the received service descriptions and the corresponding degrees to which they match the received specification of the desired service. Unfortunately, the current implementation of the PCEM component does not sort the returned list of service descriptions by matching degree.

10.6.4 PCEM Languages Processing Module

The CASCOM service coordination layer, including the Service Matchmaker Agent SMA, uses the W3C (World Wide Web Consortium) standard OWL and de-facto standard OWL-S respectively for ontology and service descriptions. Often the representation languages used internally in the several service coordination agents differ from OWL and OWL-S therefore it is necessary to perform language conversions. Since there is no agreed upon standard for describing preconditions and effects available yet, apart from the proposal of (undecidable) SWRL, the PCEM uses Prolog as its internal representation language and reasoning tool. Therefore it is necessary to convert relevant OWL and OWL-S representations involved in any service description into the component internal Prolog representations.

OWL-S service descriptions specify, among many other things such as the already mentioned preconditions and effects, the service input and output parameters and their classes. The service parameter classes are described in domain ontologies which are represented in OWL. Therefore, it is necessary to convert the OWL representations pertaining to the classes of service parameters into the component internal Prolog format. The OWL2Prolog processing mechanism of the Languages Processing module performs those conversions.

Figure 10.11 shows the Prolog internal representations generated from the OWL representation presented in Figure 10.10.

OWL-S service description language does not directly support the represen-

```
1: <owl:Class rdf:ID="RightIndexFinger">
2:   <rdfs:subClassOf rdf:resource="#RightFinger">
3:   <rdfs:subClassOf>
4:     <owl:Restriction>
5:       <owl:onProperty rdf:resource="#subPartOf">
6:       <owl:allValuesFrom rdf:resource="#RightHand">
7:     </owl:Restriction>
8:   </rdfs:subClassOf>
9: </owl:Class>
```

Figure 10.10: OWL Class Representation

```
subClassOf(rightFinger:rightIndexFinger, finger:rightFinger).
subPartOf(rightFinger:rightIndexFinger, hand:rightHand).
```

Figure 10.11: OWL Class representation in Prolog

tation of service preconditions and effects [7]. Instead, the OWL-S specification suggests that conditions (including preconditions and effects) should be represented in SWRL or in PDDL among other possibilities. Since one of the most important uses of preconditions and effects is service composition planning, the project decided to choose PDDLXML, a project brewed XML surface syntax of PDDL, because PDDL is the lingua franca of the planning algorithms used in service composition. The OWL-S/PDDLXML processing mechanism of the Languages Processing module converts PDDLXML representations of preconditions and effects into the chosen internal Prolog representations. The conversion use the OWLS2PDDL converter developed at DFKI for the OWLS-XPlan composition planner (cf. Chapter 11).

Figure 10.12 shows the Prolog internal representations of the preconditions and effects that were generated from the OWL-S/PDDLXML representations of Figure 10.1.

10.6.5 Preconditions and Effects Matching

The PCEM component performs two kinds of matching: exact matching and reasoning-based matching. Since these matching operations are done in Prolog, the module directly benefits from the Prolog built in pattern matching and reasoning capabilities. Preconditions and effects exact matching of two service descriptions (the desired service specification and the available service description) checks if the preconditions of one of the service descriptions exactly match the preconditions of the other service descriptions and if the effects of one of the service descriptions exactly match the effects of the other service description.

The exact matching of two propositions (representing either two precondi-

```
<and>
  <pred name="AvailableBook">
    <param>?Book</param>
  </pred>
  <pred name="RegisteredUser">
    <param>?IDUser</param>
  </pred>
</and>
```

(a) Preconditions

```
<and>
  <and>
    <pred name="RequestedBook">
      <param>?Book</param>
    </pred>
  </and>
  <not>
    <pred name="AvailableBook">
      <param>?Book</param>
    </pred>
  </not>
</and>
```

(b) Effect

Table 10.1: OWL-S service preconditions and effects in PDDXML

```
service(preconditions,
      [availableBook(book:BookName),
      registeredUser(number:UserID)]).
service(effect,
      [requestedBook(book:BookName),
      not(availableBook(book:BookName))]).
```

Figure 10.12: Final preconditions and effects representation in Prolog

tions or two effects) checks if there is a possibly empty variable substitution that, when applied to one or both propositions, results into two equal expressions. This operation is entirely performed by the matching operator of the Prolog language. As would be expected, reasoning-based matching is more complex than exact matching. Reasoning-based matching uses general inference rules (i.e., all deduction inference rules) and domain specific inference rules. All general inference rules are applicable to all kinds of effects and preconditions. Domain specific rules are applicable only to some preconditions or effects.

General purpose inference is performed by the built in Prolog reasoning mechanism using resolution and the closed world assumption. Domain specific inference rules are explicitly represented in Prolog and uniquely identified by rule identifiers. These rule identifiers are used to specify the domain specific rules that may be used with each precondition or effect. The representation of the domain specific inference rules and the specification of such rules that may be used with each precondition or effect integrate the domain specific knowledge.

The following example will help understand the developed reasoning algo-

rithm, when domain specific inference rules are used. The example request is a service that radiographs the client right hand index finger. The only available service in the example is a service that radiographs hands. Using conventional inputs/outputs matchmaking algorithms, or using exact matching of effects, the service description does not match the request. However the two effects will be found to match, if the matching algorithm uses the domain specific inference rule according to which, if a given service causes a certain effect on a specified object then it will cause the same effect on any of the object subparts.

The following paragraphs provide a more formal account of the way service invocations, service effects and preconditions, and domain specific inference rules are represented and used by the reasoning-based preconditions and effects matching algorithm. In the following explanations, L is a first order logic language whose terms are used to represent service invocations and whose propositions are used to represent service effects and preconditions. The relationship between services and their effects and preconditions as well as domain specific inference rules are represented in the first order language ML, which is a meta language whose terms include the propositions and terms of L.

ML possesses several predicates, among them ServiceEffect/2, ServicePrecondition/2, Class/2, and SubPartOf/2. ServiceEffect(α,ϕ) means that ϕ is an effect of the service represented by α. α is a term of L representing a service invocation. ϕ is a proposition of L representing a service effect. ServicePrecondition(α, ϕ) means that ϕ is a precondition of the service represented by α . Class(ρ,τ) means that τ is the class of ρ. ρ is a term of L, while τ is an atom of ML representing the name of a class of a given domain. Finally, SubPartOf(ρ, σ) means that σ is a subpart of ρ. ρ and σ are both terms of L.

Using ML, the relevant aspects of the available service are represented through the following expressions:

1. ServicePrecondition(XRayService(*hand*), Class(*hand*, Hand))
2. ServiceEffect(XRayService(*hand*), Radiographed(*hand*))

In these expressions, *hand* is the input parameter of the XRayService service and "Hand" is the name of a class that represents *hands*. The requested effect is represented in the expression ServiceEffect(*s*, radiographed(RightIndexFinger)) in which *s* is an uninstantiated variable representing the desired service and RightIndexFinger is a constant representing the specific finger that has to be radiographed. The informally stated domain specific inference rule is formally represented in Figure 10.13.

In the rule represented in Figure 10.13, —x/y represents the expression that is obtained by replacing x with y. Assuming the matching algorithm learns that the rule in Figure 10.13 may be applied to the effect Radiographed/1 of the x ray service, it will apply the rule replacing its variables with their specific values in the described example as follows: α = XRayService(hand), x = hand, τ = Hand, i = RightHand, ϕ = Radiographed (hand), y = RightIndexFinger, ϕ—hand/RightIndexFinger = Radiographed (RightIndexFinger).

```
ServicePrecondition(α, Class(x, τ))
ServiceEffect(α, φ)
∃i Class(i,τ) ∧ SubPartOf(i, y)
-----------------------------------
ServiceEffect(α|x/y, φ|x/y)
```

Figure 10.13: Domain Specific Inference Rule 1

```
rule(1, (serviceEffect(ReplacedService, ReplacedEffect):-
        servicePrecondition(Service, class(Object, Class)),
        serviceEffect(Service, Effect),
        class(I, Class),
        subPartOf(I, Part),
        replace(Part, Object, Effect, ReplacedEffect),
        replace(Part, Object, Service, ReplacedService))
).
```

Figure 10.14: Prolog Representation of Domain Specific Inference Rule 1

The rule premise $\exists i$ Class(i, Hand) $\wedge$ SubPartOf(i, RightIndexFinger) is satisfied since the right hand (i = RightHand) is an instance of the class Hand and the right hand index finger (RightIndexFinger) is a subpart of the right hand. Using these replacements, the instantiated conclusion of the inference rule is ServiceEffect(XRayService(RightIndexFinger), Radiographed(RightIndexFinger)), which is exactly the required effect. The rule in Figure 10.13 is translated into Prolog as shown in Figure 10.14.

The Prolog representation of domain specific inference rules uses predicate rule/2. The first argument of rule/2 is the rule identifier; and its second argument is a Prolog clause representing the rule itself. The rule conclusion is represented by the head of the clause, and the rule premises are represented by the clause body. With this design choice, the reasoning-based matching algorithm just has to assert the clauses representing domain specific rules and then rely on the Prolog built in inference mechanism to apply the rules to the desired effects and preconditions whenever necessary.

The inference rule being used in the example is not a general rule, in the sense that it cannot be applied to all preconditions and effects. For instance, it cannot be applied to a car painting service. If a car is painted blue, the car engine will not become blue, although the car engine is a subpart of the car. A given domain specific inference rule may only be applied to specified effects or preconditions. Such specifications are represented in Prolog through the predicate validation/3. The first argument of the validation/3 predicate is the service specification. The second argument is the specification of the service effect or precondition to which

```
validation(service(xRayService(hand:Hand)),
           serviceEffect(radiographed(hand:Hand),
           validRule(1)).
```

Figure 10.15: Inference rule validation clause

the rule is applicable. The third argument is the rule identifier of the applicable rule.

The validation/3 clause in Figure 10.15 states that rule number 1 may be applied to the effect radiographed(hand:Hand) of the service xRayService(hand:Hand). The hand:Hand argument means that the service input parameter Hand is of class hand.

10.6.6 Implementation

The PCEM component was developed using the OWL-S API [12], which was extended to enable processing PDDLXML conditions (in conditioned instructions), preconditions and effects for OWL-S processing; the Protege OWL-API [5] for OWL processing; and the tuProlog [9], a Java based Prolog, for the matchmaking algorithm. Prolog was chosen mainly because of its built in pattern matching and reasoning capabilities.

For results of evaluating the PCEM matchmaker, we refer to Chapter 16.

10.7 Role-Based Matchmaker ROWLS

In the following, we briefly describe the third alternative to semantic service matching used by the service matchmaker agent of the CASCOM system: the ROWLS matching component.

10.7.1 Motivation

In order to improve both the efficiency and the usability of agent-based service-oriented architectures, common organisational concepts such as social roles and types of interactions can be exploited to further characterise the context that certain semantic services can be used in.

The CASCOM abstract architecture conceives services to be delivered essentially by agents. An agent could provide an implemented Web Service by wrapping the service within an ACL interface in such a way that any agent can invoke its execution by sending the adequate (*request*) message.

However, agents are not only able to execute a service but can also engage in different types of interaction with that service. For example, in the healthcare assistance scenario, an agent providing a second opinion service should not only

be able to provide a diagnostic; it may also be required to explain it, give more details, recommend a treatment, etc.

This means that the service provider is supposed to engage in several different interactions during the provision of a service. Thus, if a physician or a patient needs one or more second opinions, they should look for agents that include those additional interaction capabilities around the "basic" *second opinion* service. In a certain sense, this approach is similar to the abstraction that an object makes by providing a set of methods to manipulate the data it encapsulates. In this case, the agent provides a set of interaction capabilities based on the service.

Taking in consideration roles and interaction types can improve the *efficiency* of the matchmaking process, for example by previously filtering out those services that are incompatible in the terms of roles and interactions.

Also the *effectiveness* of the matchmaking process can be enhanced by including information regarding roles and interactions. For instance, a diagnosis service may require symptoms and medical records as inputs and produce a report as output. However, the service functionality can be achieved either (i) by actually generating the report, (ii) by retrieving a previously done or (iii) by a brokering service to contact other (external) healthcare experts. In all the three cases the input and output are the same, but the role the service plays in the corresponding interactions is different.

10.7.2 Interaction Modelling

In order to develop role-based extensions to service matchmaking mechanisms, a subset of the RICA organisational model described in [11] and [10] was used.

Setting out from this basis, the first step was analysing different use case of the application domain scenario. For each use case, the types of social interaction as well as the roles (usually two) that take part in that interaction have been identified. The next step is an abstraction process in which the social (domain) roles/interactions are generalised into communicative roles/interactions.

The result of this analysis is a basic ontology of types of interactions and roles that take part in those interactions. Figure 10.16 shows an example, where the *SecondOpinion* interaction can be generalized in a *MedicalAdvisement* interaction and then in an *Advisement* interaction, in which the Advisor informs the Advisee about his beliefs with the aim of persuading the Advisee of the goodness of these beliefs. This ontology, and especially its generic (communicative) part, will be used in the service description and matchmaking extensions.

10.7.3 Role-Based Service Advertisements

Role-based service descriptions comprise two kinds of information related to the interactions in which the service provider agent can engage:

1. the *main role* played in the interaction, e.g. the *advisor* role in the second opinion service;

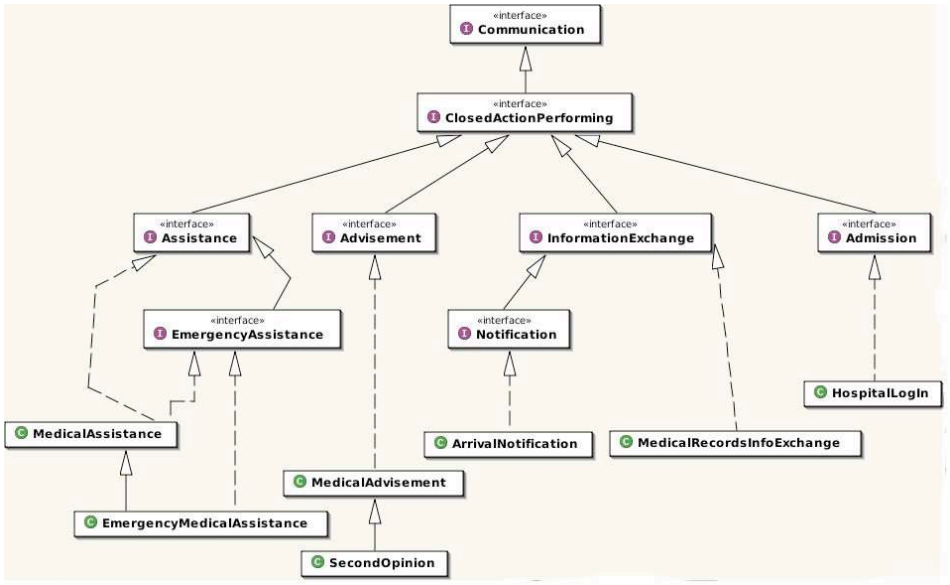

Figure 10.16: Partial interaction type ontology

Main Role	Necessary Roles
$Advisor$	$Informer$
$Explainer$	-
$Informer$	-

Table 10.2: Second Opinion role-based service advertisement

2. a set of roles that may be *necessary* to be played by the requester for the correct accomplishment of the service. For instance, in an advisement interaction of a second opinion service, the provider may need to initiate an *information exchange* interaction in which it plays the *informee* role, and the requester plays the *informer* role. Necessary roles are given by a formula in disjunctive normal form, i.e. a disjunction of conjunctions of roles.

These two fields are repeated for each main role the service can play. A service advertisement can be graphically represented by a table with two rows, in which each column contains the main role (first row) and the necessary roles (second row). Table 10.7.3 shows a role-based service advertisement for the second opinion example.

10.7.4 Role-Based Service Requests

In the case of a service requests, a query comprises two elements:

Main Roles	$Advisor \wedge Explainer$
Capabilities	$Informer, Explainer$

Table 10.3: Second Opinion role-based service request

1. *Main roles* searched. Although one role will be enough in most cases, more complex search patterns are allowed, in which the provider is able to play more than one role. As in the case of service advertisements, this expression comes as a formula in disjunctive normal form.

2. A set of roles that define the *capabilities* of the requester. These are roles the requester is able to play. This information is important if the provider requires interaction capabilities from the requesters. For example, the requester of a second opinion can inform that it is able to provide information (*informer*) if needed.

Table 10.3 shows a role-based service request for the second opinion example. The request specifies that the requester is able to play the informer and explainer roles if necessary.

Notice that this approach is compatible with services that do not make use of the role-based extensions in their description. In case a service description does not include the role-based approach, it is assumed that it has a main role *Communicator* (the top and most general concept of the ontology) and no necessary roles are required from the requester. If the request does not include a role description, it is assumed that the requester is not interested in the role-based approach and the matchmaker will omit that phase in the service matching process.

10.7.5 Role-based Service Matching Algorithm

Within the CASCOM project, a role-based matching algorithm has been developed, which takes as input a service request (R) and a service advertisement (S), and returns the degree of match (dom) between them. Essentially, it searches the role in the advertisement S that best matches the one in the query (R).

The matching algorithm is built around the matching between two roles in the taxonomy. The semantic match of two roles R_A (*advertisement*) and R_Q (*query*) is a function that depends on two factors:

1. Level of match. This is the (subsumption) relation between the two concepts (R_A, R_Q) in the ontology. A subset of the OWLS-MX filters is considered, just the same levels of match proposed in [8]:

   (a) *exact* : if $R_A = R_Q$

   (b) *plug-in* : if R_A *subsumes* R_Q

   (c) *subsumes* : if R_Q *subsumes* R_A

(d) *fail* : otherwise

2. The distance (number of arcs) between R_A and R_Q in the taxonomy.

All roles have the same importance and the generality (depth in the taxonomy) of the roles is not relevant. Both criteria are combined into a final degree of match which is a real number in the range [0, 1], so service providers can be selected by simply comparing these numbers. In this combination, the level of match always has higher priority: the value representing the degree of match is equal to 1 in case of an *exact* match, it varies between 1 and 0.5 in case of a *plug-in* match, rests between 0.5 and 0 in case of a *subsumes* match, and it is equal to 0 in case of a *fail*. Actually, any triple would work but 0.5 seems reasonable to keep the same scale in both levels (plug-in and subsumes).

There are infinite functions that fulfil that precondition. One equation that implements this behaviour is that in equation 10.1, where $\parallel R_A, R_Q \parallel$ is the distance between R_A and R_Q ($depth(R_A) - depth(R_Q)$) in the role ontology (if there is a subsumption relation between them). This kind of function guarantees that the value of a *plug-in* match is always greater than the value of a *subsumes* match, and it only considers the distance between the two concepts, rather than the total depth of the ontology tree[6], which may change depending on the domain. Furthermore, the smaller the distance between concepts (either in the case of *plug-in* or *subsumes* match), the more influence will have a change of distance in the degree of match (see Figure 10.17).

$$dom(R_A, R_Q) = \begin{cases} 1 & if\ R_A = R_Q \\ \frac{1}{2} + \frac{1}{2 \cdot e^{\parallel R_A, R_Q \parallel}} & if\ R_A\ is\ subclass\ of\ R_Q \\ \frac{1}{2} \cdot e^{\parallel R_A, R_Q \parallel} & if\ R_Q\ is\ subclass\ of\ R_A \\ 0 & otherwise \end{cases} \tag{10.1}$$

The matching algorithm compares every role in the request with the service advertisement roles, given the set of capabilities of the requester, using the aforementioned function, and returns the maximum degree of match. It uses the minimum and maximum as combination functions for the values in conjunctive and disjunctive logical expressions respectively.

10.7.6 Implementation

The role-based matchmaker was developed in Java 1.5, relying on the Mindswap OWL-S API 1.1 beta[7] for parsing OWL-S service profiles. Regarding the manage-

[6] Note that, for instance, if a linear function is used, the maximum possible distance between two concepts must be known a priori to establish the equation (e.g. $dom(x) = 1 - x/6$).

[7] `http://www.mindswap.org`

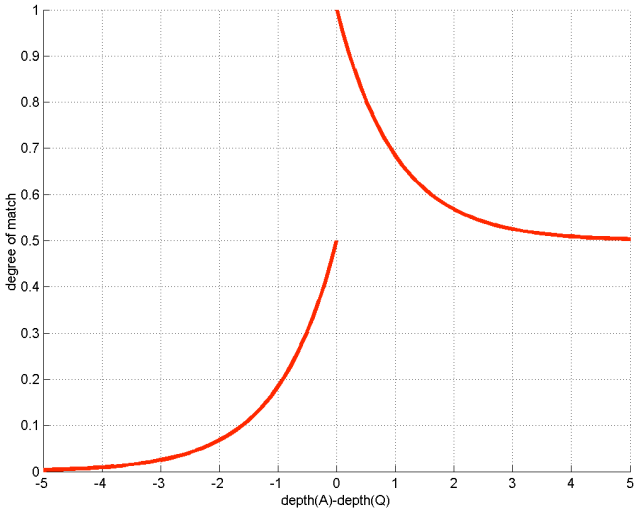

Figure 10.17: Degree of match function between two roles

ment of OWL ontologies, we adopted Jena Semantic Web Framework, a framework for building Semantic Web applications developed by the HP Labs Semantic Web research group[8].

For evaluation of the ROWLS matchmaker, we refer to Chapter 16.

10.8 Summary

The CASCOM project designed and implemented a service discovery process comprising a stage where desired services are sought and a stage where services found in the first stage are sorted according to the degree to which they satisfy specified criteria. In the first stage, services are sought according to simple selection criteria. The second stage uses more sophisticated matching criteria involving complex information processing such reasoning (e.g., subsumption) and role-based matchmaking.

The service discovery stage is carried out by the Service Discovery Agent (SDA), which looks for services in its own database and in the project distributed service directory (WSDir). In addition to seeking desired services, the SDA also registers, modifies and deletes service descriptions and service providers in its own database and in WSDir. The WSDir can be used directly without the SDA mediation. However, since WSDir is a Web Service, it does not offer an agent-based

[8] `http://jena.sourceforge.net/`

interface. If such an interface is required, the SDA should be used.

The fine grained service selection is performed by the Service Matchmaking Agent (SMA), using three different matching algorithms, each of which was implemented as a separated module integrated in the SMA architecture:

1. Hybrid input and output subsumption and information retrieval matchmaking algorithm (OWLS MX);

2. Preconditions and effects matchmaking algorithm, which performs exact, domain dependent and domain independent reasoning-based matching (PCEM); and

3. Role-based matchmaking (ROWLS).

The use of these three sophisticated matchmaking algorithms provides clear advantages in terms of both efficiency and effectiveness. For instance, role-based matchmaking may significantly reduce the number of considered services in early stages of the whole service coordination process; and reasoning-based matchmaking may identify perfectly good services, services that meet specified criteria, that would otherwise be discarded.

The three matchmaking algorithms may be combined in two distinct ways to produce the final SMA output. One of the possibilities is a sequential combination; the other is a parallel combination. In the sequential combination, each algorithm is used as a pre-filter of the next one. In the sequential combination, we have chosen to apply first the less complex algorithm (ROWLS), followed by the one with intermediate complexity (OWLS-MX), followed by the most complex of all (PCEM). This way, the more complex algorithms process fewer service descriptions. Sequential combination of the matching algorithms favors efficiency.

In the parallel combination, all algorithms are used in parallel. The results produced by each of them are aggregated in an aggregation function. Several aggregation functions (e.g., product, and minimum) may be used. Parallel combination of the algorithms favors effectiveness. Sequential or parallel combinations as well as other configuration parameters, such as the similarity threshold, are dynamically chosen during the interaction with SMA. Since these choices may be hard for SMA clients, we have predefined configurations including one that is used by default. Therefore, the SMA client has only to select one of the predefined configurations or the default one. This greatly enhances flexibility without compromising seamless interaction.

SMA and SDA interaction uses interaction protocols, agent communication language, and content language defined by FIPA; and uses ontology and service descriptions specified by the W3 Consortium. The use of standardized technologies improves interoperability and system's usability.

References

[1] A. Bernstein and C. Kiefer: Imprecise RDQL: Towards Generic Retrieval in Ontologies Using Similarity Joins. Proc. ACM Symposium on Applied Computing, Dijon, France, ACM Press, 2006.

[2] I. Constantinescu and B. Faltings: Efficient matchmaking and directory services. Proceedings of IEEE/WIC International Conference on Web Intelligence. 2003.

[3] A. Fernández, M. Vasirani, C. Cáceres and S. Ossowski: A Role-Based Support Mechanism for Service Description and Discovery. Service-Oriented Computing: Agents, Semantics, and Engineering. LNCS, 4504, pp. 132–146, Springer, 2007.

[4] M. Klusch, B. Fries and K. Sycara: Automated Semantic Web Service Discovery with OWLS-MX. Proceedings of 5th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2006); Hakodate; Japan; ACM Press. 2006.

[5] H. Knublauch et al.: Protege-OWL API. Available online at http://protege.stanford.edu/. September 21, 2006.

[6] L. Li and I. Horrocks: A software framework for matchmaking based on semantic web technology. Proceedings of the twelfth international conference on World Wide Web, pages 331-339. ACM Press. 2003.

[7] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara: OWL-S 1.1 Release; http://www.daml.org/services/owl-s/1.1/overview/. 2004.

[8] M. Paolucci, T. Kawamura, T. Payne and K. Sycara: Semantic matching of Web Services capabilities. In Proceedings of the First International Semantic Web Conference on The Semantic Web, Springer-Verlag (2002) 333-347

[9] E. Denti, A. Omicini and A. Ricci: tuProlog: A Light-weight Prolog for Internet Applications and Infrastructures. Proceedings of the 3rd International Symposium on Practical Aspects of Declarative Languages (PADL 2001); Las Vegas; NV; USA; 11-12; LNCS 1990, Springer-Verlag, 2001.

[10] J. M. Serrano and S. Ossowski: A compositional framework for the specification of interaction protocols in multiagent organizations. Web Intelligence and Agent Systems: An international Journal. IOS Press. 2006.

[11] J. M. Serrano, S. Ossowski and A. Fernández: The Pragmatics of Software Agents - Analysis and Design of Agent Communication Languages. Intelligent Information Agents - The European AgentLink (Klusch et al. ed.), pp 234-274, Springer. 2002.

[12] E. Sirin and B. Parsia: The OWL-S Java API. Proceedings of the Third International Semantic Web Conference. 2004.

[13] K. Sycara, S. Widoff, M. Klusch and J. Lu: LARKS: Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace; Journal of Autonomous Agents and Multiagent Systems. Kluwer Academic Press. 2002.