Service Management in Open Environments

Author: Elena del Val Noguera
Supervised by: Dr. Miguel Rebollo Pedruelo

Dissertation submitted in partial fulfillment
of the requirements for the Master in
Artificial Intelligence, Pattern Recognition and Digital Image
Valencia

September, 12, 2008

Contents

List of Figures

List of Tables

Chapter 1

Introduction

One of the toughest jobs for managers today is keeping up with the rapid changes in technology. The advent of service-oriented architectures makes this more important, because these technologies are changing the way of building internal systems and how internal systems interact with external systems. The aim of the service-oriented architectures is build compatible software components that will reduce costs of software systems and at the same time increasing the capabilities of the systems.

SOA is an architectural style whose goal is to achieve loose coupling among interacting software agents. A service is a unit of work done by a service provider to achieve desired end results for a service consumer. Both provider and consumer are roles played by software agents on behalf of their owners. In SOA, web services are a step along a much longer road. The notion of a service is an integral part of component thinking, and it is clear that distributed architectures were early attempts to implement service-oriented architecture. It is important to recognize that web services are part of the wider picture that is SOA. The web service is the programmatic interface to a capability that is in conformance with protocols. So web services provide us with certain architectural features and benefits specifically platform independence, loose coupling, self description, and discovery and they can enable a formal separation between the provider and consumer because of the formality of the interface. In fact web services are not a mandatory component of a SOA, although increasingly they will become so.

Service-Orientation supports a common principles that are commonly accepted by all major SOA platforms in the SOA industry. The following guiding principles define the ground rules for development, maintenance, and usage of the SOA [22]:

- Reuse, granularity, modularity, composability, componentization, portability and interoperability
- Compliance to standards (both common and industry-specific)

- Services identification and categorization, provisioning and delivery, and monitoring and tracking

The following specific architectural principles for design and service definition focus on specific themes that influence the intrinsic behavior of a system and the style of its design [9]:

- Service encapsulation: Many web-services are consolidated to be used under the SOA Architecture. Often such services have not been planned to be under SOA.
- Service loose coupling: Services maintain a relationship that minimizes dependencies and only requires that they maintain an awareness of each other
- Service contract: Services adhere to a communications agreement, as defined collectively by one or more service description documents
- Service abstraction: Beyond what is described in the service contract, services hide logic from the outside world
- Service reusability: Logic is divided into services with the intention of promoting reuse
- Service composability: Collections of services can be coordinated and assembled to form composite services
- Service autonomy: Services have control over the logic they encapsulate
- Service optimization: All else equal, high-quality services are generally considered preferable to low-quality ones
- Service discoverability: Services are designed to be outwardly descriptive so that they can be found and assessed via available discovery mechanisms

Web services naturally support a subset of these principles, which provides an indication as to why the web services technology platform is considered so suitable for building service-oriented solutions.

Nowadays, there are two of the principles presented before that are getting more relevance: discovery and composition. Reusable, loosely coupled services can be achieved, but even the most reusable service is not useful if it cannot be found by those responsible for creating potential consumers. Furthermore, even the most loosely coupled services will have limited reuse potential if they cannot be assembled into effective compositions. This is where the principles of service discoverability and service composition became important.

The characteristic of discoverability essentially helps avoid the creation of redundant services or services that implement redundant logic. Because each service operation provides a potentially reusable piece of automation logic, the metadata attached to a service needs to sufficiently describe not only the service's purpose, but also the functionality offered by its individual operations.

As the number of services grows in size, service compositions will become an unavoidable and increasingly important design aspect of building service-oriented solutions. The main reason this particular principle is so important is because it ensures that services are designed in such a manner so that they can participate as effective members, or controllers, of these compositions.

The requirement for any service to be composable also places an emphasis on the design of service operations. Composability is simply another form of reuse and therefore operations need to be designed in a standardized manner (and with an appropriate level of granularity) to maximize composition opportunities.

Discovery and service composition principles are closely related. A fundamental rule of service abstraction is that a service can represent any range of logic from any types of supported sources, including other services. If services encapsulate others, we have a composition. To build a useful composition, the service designer will need to find the most suitable services to act as composition members. Furthermore, once the composition is completed and deployed, potential consumers of the service representing the composition will benefit from an awareness of its existence, purpose, and capabilities.

In many cases the problem in service discovery and composition is that there are many available services which provide similar or identical functionality, although with different Quality of Service (QoS), a choice needs to be made to determine which services are to participate in a composite service. One parameter that could be considered is time. Sometimes service quality decays with time and, in some way, the provider have to determine how much time takes the service to be executed. If the execution of the service takes to much time, maybe the service has not interest for the user.

This problem is present in open environments where entities like web services or agents need to locate other entities to achieve cooperation, delegation or interoperation. Web services and agents are autonomous entities that provides services to others, maintaining their independency and modularity. In both cases they can cooperate to develop complex tasks. But agents are above services, providing some solutions to the drawbacks of web services or even sematic web services. The nature of agents, as intelligent and flexible entities with autoorganizative capabilities, facilitates the implementation automatic service composition and discovery.

1.1 Objectives

The work presented is inside the projects THOMAS (MeTHods, Techniques and Tools for Open Multi-Agent Systems)[1] and Consolider Agreements Technologies[2]. The aim of THOMAS is the investigation and development of dynamic agent organizations that self-adjust in order to make the most of their current environment. These organizations could appear in dynamic or emerging societies of agents such as Grid domains, peer-to-peer networks, or other environments

[1]Project TIN2006-14630-C03-01

[2]Project CONSOLIDER-INGENIO 2010 under grant CSD2007-00022

in which the agents coordinate in a dynamic way in order to offer composite services. The aim of Agreement Technologies is developing models, frameworks, methods and algorithms for constructing large-scale open distributed computer systems. The main idea is anticipate solutions for the needs of next generation computing systems where autonomy, interaction and mobility will be the key issues. In both projects there is an objective in the development of intelligent service coordination techniques/methods within open, decentralized multiagent systems: intelligent service location (directory services, syntactic and semantic comparison techniques for services) and generation and adaptation of composed services. This work is included inside this objective an also has a more specific objectives:

- Analyze the service discovery and composition problem
- Review the solutions to discovery and composition problem in web services and in agent systems
- Analyze the weak points in the solutions to discovery and composition
- Propose an extension of OWL-S for service composition
- Integrate the techniques in service discovery and composition in the THOMAS Service Facilitator module which is responsible of service discovery and composition.

1.2 Organization

This work is organized in five chapters. The work starts with a review of the current situation in service discovery and composition area and ends with a service composition proposal.

- Chapter 2: In this chapter a comprehensive survey in automatic service discovery is presented. We have scrutinized the literature and critically reviewed works originating from the fields of web services and agents to provide a comprehensive overview of service discovery work to date. We start by defining the main elements that take part in the service discovery process and the different stages that are part of the process. To continue, we review algorithms in the field of web services taking into account their features such as the information that use in their process, if they manage different ontologies or if they take into account service composition. In the next section we review some agent features that could be useful to consider in the discovery process and present some works in this area. Furthermore, we present a classification of some of the algorithms presented in agent systems according to the environments where they are applied. Finally, we make some final conclusions and remarks.

- Chapter 3: In this chapter a service oriented architecture is presented. THOMAS is an open multi-agent system architecture consisting of a related set of modules that are suitable for the development of systems applied in open environments. THOMAS is an extension of FIPA architecture, it expands its capabilities to deal with organizations, and to boost up its services abilities. The modules that compound the architecture are presented.

- Chapter 4: In this chapter the Service Facilitator(SF) module is presented and described with more detail. The SF is a redefinition of the FIPA Directory Facilitator which is able to deal with services in a more elaborated way, following Service Oriented Architectures guidelines. A SF view from an high-level design and from a low level-design is provided. Furthermore a proposal included in the fuctionality of the SF for service composition is presented. This proposal is an OWL-S extension to include temporal qualified services by including duration as a non-functional parameter and temporal constraints in the preconditions and effects of the service. The temporal annotated services in OWL-S are translated in durative actions in PDDL 2.1, so any planner that deals with that language can be used.

- Chapter 5: An example of a Packing Cell is presented. This problem is modelled with THOMAS and it is an explanation with more detail of the service composition process presented in the previous chapter.

- Chapter 6: Conclusions and future work.

Chapter 2

Service Discovery and Composition

2.1 Introduction

Nowadays, the need for communication among loosely-coupled distributed systems is bigger than ever. In open, dynamic and distributed environments, agents and web services offer a variety of services that can be discovered taking into account certain kind of information in order to achieve a goal or to give a response to a necessity. The main obstacle affecting the service discovery mechanisms is heterogeneity between services. The identification of different kinds of heterogeneity gives an impression on what has to be considered in order to avoid or mitigate them [56]: technological, ontological and pragmatic heterogeneities.

There is an enormous amount of diverse work originating form different communities who try to overcome different aspects of this heterogeneity in order to match the best service available and claim some sort of relevance to service discovery [74][73]. Web services and agents are two important fields that face the problems in a different way but also have some points in common. This chapter aims to make a comprehensive survey in automatic service discovery and composition. We have scrutinized the literature and critically reviewed works originating from the fields of web services and agents to provide a comprehensive overview of service discovery and composition work to date.

We start in section 2.2 by defining the main elements that take part in the service discovery and composition process and the different stages that are part of the process. To continue, in section 2.3 we review algorithms in the field of web services taking into account their features such as the information that use in their process, if they manage different ontologies or if they take into account service composition. In the section 2.4 we review some agent features that could be useful to consider in the discovery process and present some works in this area. Furthermore, in section 2.4.2 we present a classification of some of the algorithms presented in agent systems according to the environments where they

are applied.

2.2 Discovery and Composition process

Discovery and composition process aim is to locate and compose, if it is necessary, services with similar or exactly service descriptions. Mainly, this process consists of possible matchings between possible services providers and service requests. It could be structured in two stages: (i) a first stage where the possible providers that offer the requested capability are selected using a matchmaking algorithm, and (ii) a second stage that refines the set of providers. A third stage where the obtained results in previous stages are evaluated can be also considered (Fig2.1). More detailed description of these stages is presented in the following items.

- *Selection Process.* In this stage a service request is received and sent to a matchmaker. The matchmaker is responsible of finding a suitable service or set of services according with the requested description. In this stage, the matchmaker uses a matchmaking algorithm which returns a service or set of services, ordered according to the suitability with the request. The suitability depends on the information that the algorithm considers, usually is the degree of similarity between service provider and server request and it will depend on the degree of similarity between input and output parameters (IO' s).

- *Ranking.* In this stage the set of possible service providers is refined according to additional information that the client has defined previously to choose the more suitable provider. To achieve this, the concept of non-functional attributes, in most of cases related to quality of service (QoS), is introduced. These attributes are used in ranking functions that gives a mark or punctuation to the set of providers selected in the previous stage. For example we can consider a set of candidate services S and a ranking function $F(S, x, R) = S'$, where S is the set of services, x is the ranking attribute and R the order (ascendent, descendent). The result of the ranking function is the ordered set of candidate services. If we consider a set of services $S = \{s1, s2\}$, $x = \{cost\}$ and $R = \{ascending\}$, assuming that Cost of $s1$ is more than $s2$, after the execution of the ranking function we obtain S' $\{s2, s1\}$. Furthermore, the user can define an threshold to filter the services obtained.

- *Evaluation.* When the results of the matchmaking process are obtained, it is recommended evaluate these results to identify possible modifications in some matchmaking parameters. The most important points to be evaluated are the quality of the results and the system execution. Obviously, a measure for these aspects is needed. In these cases, precision and recall measures are used. Recall measures how good discovery architecture retrieves all relevant services. Precision measures how good the architecture retrieves relevant services. A service can be considered relevant if the

user that performs the query consider it relevant. If we consider x as the number of relevant services retrieved for the user, n as the total number of relevant available services and N as the total number of services returned to the user. Therefore, we can define recall as x/n and precision as x/N. An evaluation of information retrieval methodologies and a survey of the current evaluation approaches in the area of semantic services discovery is presented by Kuster et al. [42].

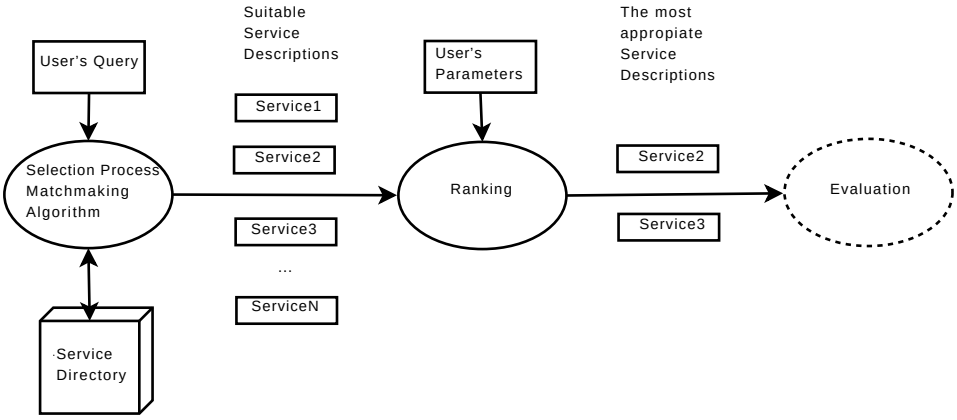

Figure 2.1: Discovery Process

2.3 Semantic Web Services Discovery

Since the appearance of semantic web, web service discovery has been changed. At the beginning, all existing approaches used syntactic similarity to establish the degree of matching between two service descriptions. With the arrival of semantic web, service descriptions include data structures and relationships between other concepts. Furthermore, constraints and rules that allows inference process are present. In this situation, more complex algorithms which have to consider semantic information to attend user requirements are needed. These algorithms are based on semantics. Moreover, in some complex environment, in which several self-interested entities are involved, it can be possible to establish discovery models based on negotiation protocols. In this section the main approaches of the web service discovery are defined. In each approach we present some of the existent discovery algorithms and its main features.

In following sections the main approaches which are in service discovery are reviewed. Basically these approaches can be divided in three groups: *discovery based on keywords or syntactic information* which is commonly used in web service environments, *discovery based on semantics* that is present in environments of semantic web services and finally *discovery based on negotiation.*

2.3.1 Service discovery based on keywords.

It is a first step towards semantic service discovery. Most of the times, services can be filtered by doing a search based on keywords. A typical keyword scenario is a search engine that receives queries with keywords and after that the engine engages query keywords with service description keywords. This model is followed by the legacy UDDI Standard and the discovery mechanism it supports. The retrieval stage comprises a user or a search program, entering a query to the catalogue. The query consists of keywords, which are matched against the stored descriptions. The matched web services are then returned as a candidate answer set and the user browses them in order to find which one of them really suits her needs. The available search tools are very simple and do not take into consideration any cross-correlations between web services and the qualitative characteristics of each web service, forcing the user to repeat the search from the beginning using new key words. The main problem of keyword discovery is that do not allow retrieval of WS with similar functionality; two WSDL descriptions can be used to describe the same service but with different words. A web service discovery review based on keywords is presented in [5].

2.3.2 Service discovery based on semantics.

The use of vocabularies with a formal and explicit semantic is considered as a second approach. Ontologies, which provide a formal and explicit specification of shared concepts, can be used for this goal. Ontologies give us a shared and explicit terminology to describe web services and queries with a logic formalization. By maintaining ontology hierarchies, it is possible to perform semantic matching, which is subsequently performed by exploiting the subsumption capabilities of ontology languages. Semantic matching has several advantages. First of all it provides matching flexibility, because results are returned can differ syntactically with the input query. It also provides accuracy since no matching is performed unless this is derived from the hierarchy and finally the concept of matching degree can be supported [27].

In this kind of discovery, some approaches according to the service description language used have been found. Among them, the most important ones use OWL-S, WSMO and SAWSDL (based on WSDL-S) languages. These languages provide the answer to the main questions that arise when a web service has to be described: what are the service requirements from the user?, What is provided by the service to the users?, How does the service work?, and How a service can be used?.

Taking into account this information, there are different matchmaking proposals in the area of web services and semantic web. Some of them have been reviewed paying attention to features such as the information that they use to deal with the matchmaking process, if they take into account service composition or cross ontologies or if they use QoS information during the discovery process. According to that, the approaches reviewed have been classified in the following types:

- *IOPE's*: Inputs, Outputs, Preconditions and Effects are used during the discovery process.
- *Composition*: when there is not a single service which can answer the user query the algorithm for discovery tries to give an answer composing single services.
- *Cross ontologies*: the services and user queries can be described using different ontologies. This is an important point to take into account in the process of discovery.
- *Hypergraphs*: the structure used to represent service information is a hypergraph that facilitate also the process of discovery.
- *Model checking*: Techniques commonly used in Model checking can be also be considered in web services composition.
- *Non-functional parameters*: the algorithm besides the IOPE's tales into account parameters related with QoS (Quality of Service).
- *Efficient matching*: discovery process is a non-trivial problem. Some algorithms use different ideas to improve the efficient of the discovery process.
- *Other trends*: there are proposals that present other techniques (reasoning techniques or fuzzy logic) that can be also taken into account.

IOPE's

In general, most of the existing matchmaking algorithms use just inputs and outputs as information to determine the matching degree between requests and announcements. Most of these algorithms are based on Paolucci's algorithm[58]. This algorithm considers that a matching between a service advertisement and a service request consists of matching all the service request outputs with those of the service advertisement; and all the inputs of the service advertisement with those of the service request [71]. The degree of similarity between service provider and server request will depend on the degree of similarity between input and output parameters (IO' s). The similarity degree depends on the relation between the concepts (taken from the ontologies) that are being compared, and it is reduced to calculate the minimal distance between them in the taxonomic tree. The denomination of degrees varies according to literature [3][43][58][18]. Mainly, can be stablished the following degrees of matching:

- *Exact*: when the two concepts in the request and in the advertisement are equivalent.
- *Subclass of*: when request concepts are a subclass of the advertisement concepts.
- *Subsumption*: there are two types:

  - Plug-in or Contained: when the concepts of the advertisement A include the concepts of the request P. Formally, $P \subseteq A$. In this case, the request can be satisfied due to the advertisement concepts are more general than the requested concepts. Therefore, it exists the possibility that the client can achieve the goals. But it is not considered that a directed subclass relation exists.
  - Subsume or Container: when the request concepts include the advertisement concepts; formally, $A \subseteq P$. This kind of matching do not satisfy completely the request but it can be considered as a valid, partial solution because it allows the client to achieve partial objectives or goals.

- *Fail, Null o Disjoint*: when there is no inclusion relation between concepts; formally $(A \bigcap P) \subseteq \perp$.

Once that the similarity degree is calculated for each IO, the next step is ordering all the services. To order them, an algorithm that orders the services obtained after the matchmaking process is designed. This algorithm gives more priority to the service output matchings than to the service input matchings due to what the client hopes to obtain (the output of the service) is more important. Given two services, first they are ordered according to the outputs and only if there is a draw the services will be ordered with the similarity degree of the inputs.

Wolf-Tilo defines a different matchmaking algorithm [77]. In this algorithm, first of all they make a search based on keywords and, later, once is obtained a list of results, the user could introduce IO's parameters that must have the services and the values for these parameters. Finally, a reasoner eliminates those services that do not have these defined parameters and the resultants will be executed with the values that were introduced by the client. The results of the executions are ordered following some constraints that benefit the client, as the quality of service.

Klusch presents OWLS-MX, a hybrid matchmaking algorithm that computes the degree of semantic matchmaking for a given pair of service advertisement and request by successively applying five different filters: *exact*, *plug-in*, *subsumes*, *subsumed-by* and *nearest-neighbor*[39]. The first three are logic based only whereas the last two are hybrid due to the required additional computation of syntactic similarity values. The objective of hybrid semantic web service matching is to improve semantic service retrieval performance by appropriately exploiting means of crisp logic based and approximated semantic matching.

The OWLS-MX matchmaker takes any OWL-S service as a query and returns an ordered set of relevant services that match the query, each one annotated with its individual degree of matching, and syntactic similarity value. The user can specify the desired degree and syntactic similarity threshold. OWLS-MX determine the degree of logical match and the syntactic similarity between the conjunctive I/O concept expressions in OWL-Lite. 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

Algorithm 1 Generic OWLS-MX Match: Find advertised services S that best hybridly match with a given request R; returns set of (S, $degreeOfMatch$, $SIM_{IR}(R, S)$) with maximum degree of match (dom) unequal FAIL, and syntactic similarity value exceeding a given threshold α

1: **function** MATCH($Request$ R, α)
2: **local** $result$,$degreeOfMatch$,$hybridFilters$={SUBSUMED-BY, NEAREST NEIGHBOUR}
3: **for all** (S,dom) $\in$ $CANDIDATES_{inputset}$ ($INPUTS_R$) $\wedge$ (S, dom') $\in$ $CANDIDATES_{outputset}$($OUTPUTS_R$) **do**
4: $degreeOfMatch \leftarrow$ MIN(dom, dom')
5: **if** $degreeOfMatch \geq minDegree \wedge (degreeOfMatch \notin hybridFilters \vee SIM_{IR}(R,S) \geq \alpha)$**then**
6: $result$:= $result \cup \{ (S, degreeOfMatch, SIM_{IR}(R,S))\}$
7: **end if**
8: **end for**
9: **return** ($result$)
10: **end function**

given similarity threshold. The pseudo-code of the generic OWLS-MX matching process is shown in algorithm 1.

Composition

Another aspect that web service discovery considers is that the discovery of services would be seriously limited to discover single services if the algorithms do not address the issue of discovering service compositions. In many situations, queries that cannot be satisfied by a single service might be frequently satisfied by composing several services. Currently, this feature is present in some discovery algorithms [4][10][15][38][11] [48][34][18]that achieve a more flexible matching. Most of them use the information provided by the service model to achieve this goal. In these section, algorithms with more flexible matching for service composition are presented.

Aversano displays a discovery algorithm that allows service composition discovery. Their algorithm analyzes DAML-S service profiles as the previous algorithms. The algorithm takes as objective the outputs of the user request and considers the possibility of reaching it with only one service. If to satisfy the user request with only one service is not possible, the method includes a backward-chaining algorithm with the purpose of verifying the possibility of finding a match for the user request by means of a composition of several services. This algorithm is also capable of performing cross ontology matching for service descriptions that use different ontologies. However, it crosses ontologies at query time, hence severely affecting the efficiency of the whole procedure. In future work, the algorithm will be extended to take into account pre-conditions and post-conditions [4].

The matching algorithms described until now are based on DAML-S/OWL-S and use the service profile. The matching based on the service profile shows up some problems when composition is needed. This process is similar somehow to matching two black boxes: a service request asking for two outputs O_1 and O_2 with a service advertisement that provides either O_1 or O_2 but not necessarily both (e.g., a choice process)[15]. This match would not be correct because the process is not capable of providing these two outputs. In order to clearly

specify the behavior of such service, two service profiles, corresponding to the two alternatives, have to be provided. This would lead to advertise a large number of profiles. Moreover, analyzing web services only through their service profile (i.e., their IOs) can severely affect the process of discovery of service aggregations that satisfy a request. That is because the service profile does not describe the internal behavior of services so, in some cases, it does not provide valuable information needed for composing services. For these reasons, there are many discovery algorithms that use the information, provided by the process model.

The first discovery algorithm based on the analysis of the DAML-S Process Model was proposed by Bansal and Vidal [10]. The Bansal and Vidal algorithm stores advertisements of services as tree structures corresponding to their process models. The compound processes correspond with intermediate nodes, whereas the atomic processes correspond with the leaves. The root of the process model corresponds with the root of the tree. The matchmaking algorithm begins in the root of the tree of the advertisement of the service and, recursively, it visits all the subtrees finishing in the leaves. For each node, the corresponding matchmaking algorithm verifies the compatibility between the IOs of the service and the IOs of the query.

Service Aggregation Matchmaking (SAM)[15] is an extension of the matchmaking algorithm proposed by Bansal and Vidal that provides more flexible matching and considers service composition in the situations in which the queries cannot be satisfied only by one service [10]. SAM can also return, when a complete matchmaking is not possible, a list of partial matchings (a composition of subservices that can provide only certain requested outputs by the client). Besides, when it does not find any match, SAM is able to suggest to the user additional inputs that can be enough to reach complete match. The main lack of this algorithm is that does not consider the use of different ontologies.

The SAM algorithm consists of two main parts:

- Construction of a graph representing the dependencies among atomic processes of the services in the registry;
- Analysis of such dependency graph to determine a service composition capable to satisfy the query (or part of it, when no service composition can fully satisfy the query).

For atomic nodes for example, Match checks whether the corresponding atomic process is already contained in the graph. If this is not the case, Match verifies the compatibility between the inputs and the outputs of the atomic node and the data nodes currently contained in the graph. If all its inputs or at least one of its outputs are contained (w.r.t. compatibility) in the graph then the atomic process is considered to be matched and added to the graph. Match then creates a new process node, new data nodes and all needed edges and constraints, and inserts them in the dependency graph (See Alg.2).

Klus presents an algorithm based on hypergraphs [38]. This algorithm also deals with the goal of service composition. In section 2.3.2 of hypergraphs the

Algorithm 2 MATCH

```
1: Match(AtomicProcess P, Graph G)
2: if (P ∉ G) then
3: if (I_p ∈ G ∨ O_p ∪ G ≠ ∅) then
4: Add P to G;
5: forall outputs O in O_p do
6: if (O ∉ G) then Add O to G;
7: Add (P,O) to G;
8: forall inputs I in I_p do
9: if (I ∉ G) then Add I to G;
10: Add (I,P) to G;
11: forall predecessors PR in Prev_p do
12: if (PR ∈ G) then Add (PR, P) to G;
13: forall choice processes PC in Choice_p do
14: if (PC ∈ G) then Add (P, PC) ∨ (PC, P) to G;
```

main algorithm features will be explained.

Another interesting point to take into account related with service composition is related with goals. There are languages, such as WSMO, that consider goals to achieve a composition. The work presented by Birna and Wirsing points out how goal-oriented techniques, which increase flexibility in handling failures, can be applied in the context of service-oriented systems and, specifically, in web services composition [75].

Cross Ontologies

Crossing ontologies is another issue that service discovery algorithms have to face. Individual users or user communities hope to be able of making queries about interesting services using descriptions that are expressed in terms of their own ontologies, which do not have to fit in with the ontologies used in the searched service descriptions. There are algorithms that do not address properly or do not address at all the problem of crossing ontologies. In some of them, even all services are assumed to share the same ontology or multiple ontologies are inefficiently crossed. This feature is getting more and more important to facilitate semantic interoperability between services [17][4][59][13][39].

The algorithm proposed by Cardoso allows to manage multiple ontologies. The algorithm uses a function to calculate the degree of similarity of two concepts taking into consideration the ontology(ies) associated with the concepts, being compared [17]. Different similarity functions are used depending on the concepts, if they are from the same ontology or from distinct ontologies. The evaluation of the similarity of two concepts is based on their composing properties. The similarity function, of the concepts which belongs to the same ontology, computes the geometric distance between the similarity of the domains of concept O and concept I and the ratio of matched input properties from the concept I. The similarity function for the concepts which do not share a common ontology uses the same rationale that have been exploited to compare input and output concepts from the same ontology without any parent/child relationship. Additionally, they also take into account syntactic similarities among concepts.

Pathak also deals with the fact that different users may use different ontolo-

gies to specify the desired functionalities and capabilities of a service [59]. An ontology mapping during service discovery is proposed. Such terms and concepts in the service requester's ontologies are brought into correspondence with the service provider's ontologies. To do the mappings they use interoperability constraints, i.e. a set of relationships that exists between elements from two different hierarchies.

An extension of SAM based on hypergraphs that allows to cross different ontologies is presented by Brogi et al. [13]. The matchmaking system consists of two main modules: the *Hypergraph Builder* and the *Query Solver*. The *Hypergraph Builder* analyses the ontology-based descriptions of the registry-published services in order to build a labeled directed hypergraph, which synthesizes all the data dependencies of the advertised services. The vertexes of the hypergraph correspond to the concepts defined in the ontologies employed by the analyzed service descriptions, while the hyperedges represent relationships among such concepts (*subConceptOf*, *equivalentConceptOf* and *intra-service dependency*). The *Query Solver* explores the hypergraph by suitably considering the *intra-service* and *inter-service* dependencies to address the discovery of (compositions of) services as well as by considering the *subConceptOf* and *equivalentConceptOf* relationships to cope with different ontologies. There are other proposals of matchmaking algorithms based on hypergraphs but they do not consider the task of crossing ontologies. We will see these algorithms in next section.

Hypergraphs

In order to store the knowledge derived from the preprocessing of ontologies and service descriptions, a data structure capable of suitably representing service and data dependencies is needed. Hypergraphs seem to be a good candidates as dependencies can be naturally modelled by means of hyperedges. A hypergraph is a generalization of a graph, where edges (hyperedges) can connect any number of vertexes. Formally, a hypergraph is a pair (X, E) where X is a set of nodes or vertexes and E is a set of non-empty subsets of X called hyperedges. While graph edges are pairs of nodes, hyperedges joints arbitrary sets of nodes and they can, therefore, contain an arbitrary number of nodes(Fig2.2). There are many proposals in discovery algorithms that use hypergraphs to manage the data provided by service descriptions and requests.

Brogi et al. present a directed hypergraph that is used to model the necessary data to discover and compose suitable services to answer a user request [14]. In this approach, the vertexes of the hypergraph correspond to the concepts defined in the ontologies used by the analyzed service descriptions, while hyperedges represents relationships among such concepts. The relationships could be: *subConceptOf*, *equivalentConceptOf*, *intra-service dependency*.

The discovery algorithm takes as input a client query specifying the set of inputs and outputs of the desired service (composition). The search algorithm explores the hypergraph by performing a depth-first visit, in order to discover the (compositions of) services capable of satisfying the client request. Basically, the algorithm searches the services which produce each requested output. If

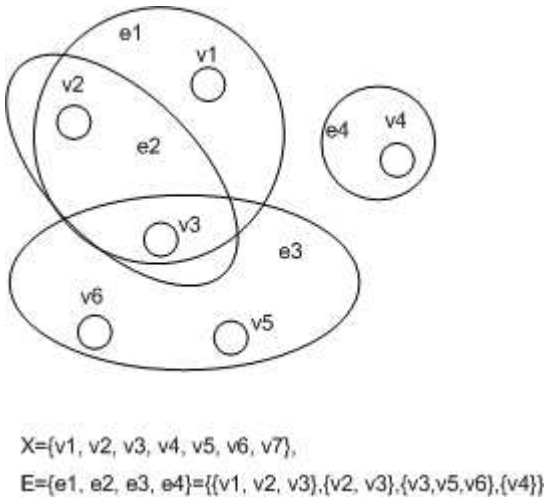

Figure 2.2: Hypergraph

the algorithm does not find a service which produce the output, the algorithm fails since the query can not be fulfilled. Otherwise, for each service which generates a requested output, the algorithm adds that service to *composition* and updates the set of *avaliableOutput* by adding the service outputs and updates *neededOutput* by adding the inputs of the service and by removing the concepts that are now available. Next, the algorithm continues recursively.

Prabhu uses hypergraphs to model web services more accurately than regular graphs [61]. The graph G is the search space, composed of vertices V representing concepts, and directed edges E representing web services in W (set of existing web services that are available to the central agent). The central composition algorithm works both forwards from seed I (set of user given input concepts) and backwards from seed O (set of user expected output concepts) simultaneously, performing a breadth-first search from each end (albeit differently) on G. At the end of each stage (BFS tree layer), it checks if there is a collision between the two trees. When a collision is detected, it signifies the shortest path calculation.

Bailey presents a method for discovery interesting collections of web services, according to user specified cost constraints [6]. To carry out this task they reduce the discovery problem to one involving hypergraphs. Each web service is modeled as a vertex, W represents the set of all web services and w represent a single vertex. Each edge corresponds to a set of vertices (web services) which offer a particular functionality requested by the user. F is used to represent the set of all possible functionalities and f to represent a single functionality. A transversal of the web service hypergraph now corresponds to a set of web services that cover all the functionalities requested by the user. The process to find sets of web services satisfying the user query can be reduced to finding

transversals of the web service hypergraph.

Benatallah et al. propose a service discovery algorithm derived from hypergraphs theory. They show that the discovery problem is similar to the so-called best covering problem [11]. The discovery problem is viewed as a service request Q and a ontology of services T. The goal is to compute the best combinations of web services that satisfy as much as possible the outputs of the request Q and that require as little as possible of inputs that are not provided in the description of Q. The best profile covering problem can be interpreted in the framework of hypergraphs as the problem of finding the minimal transversals with a minimal cost.

Yang et al. use arc-labeled and arc-weighted trees to represent product/service requirements and offers. An arc-labeled, arc-weighted tree is a 5-tuple $T = (V, E, Lv, Le, Lw)$ where V is a set of nodes, E a set of arcs, Lv is a set of node labels, Le a set of arc labels and $Lw = [0, 1]$ is a set of arc weights. (V, E, Lv, Le) is an arc-labeled tree and there is an $(n \rightarrow 1, n \geq 1)$ mapping from the elements in E to the elements in Lw [79].

A tree similarity algorithm which traverses input trees top-down (root-leaf) and then computes their similarity bottom-up is proposed. During tree similarity computation, when a subtree in T_1 is missing in tree T_2 (or viceversa), the algorithm computes the simplicity of the missing subtree. The tree simplicity measure takes into account the node degree at each level, the depth of the leaf node and the arc weights for the recursive tree simplicity computation. This algorithm allows partial product descriptions representations via subtrees missing. In order to take into account the effect of a missing subtree on the similarity between two trees, the algorithm uses a implicit measure.

Hashemian uses an specific notation, called interface automaton—which is a state-base model—in order to formally model web services [34]. The information that an interface automaton exposes is the IO of a component and the temporal ordering of the actions it performs. This information can be extracted form the OWL-S specification of web services. Based on the properties exposed by interface automaton of each web service, three pieces of information are stored in a repository: its set of inputs, its set of outputs and dependency information between IO's of the web service. The repository is stored as a graph that contains web services information. The nodes represents I/O and there is a directed edge form node v_1 to node v_2 if and only if there is a dependency between the input and the output. The problem is solved in two steps: (i) finding web services that can potentially participate in the composition, and (ii) finding the composition setup based on the web services found in the previous step. In the first step, a BFS (Breadth First Search) procedure finds all dependencies that are satisfied by the dependency graph. In the second step, binary composition operators supported by OWL-S (composition, sequential execution, conditional execution and parallel execution) are used.

Model Checking

Web services are composed online from pieces of software created by different programmers. Individual services can be checked to ensure that they are error-free, but when new services are composed there are no means to check whether the composed service fulfils its purpose. Some formal methods, as model checking, has been proposed to verify the correctness of complex services. But current languages are semiformal, so the correctness of the composition depends on the cleverness of the designer. To use formal models requires translations from the languages used to describe WS, such as BLEP or WSDCL, into more formal ones.

Gao et al. translate web services specified in BPEL4WS into pi-calculus, which is nearer to programming languages than finite automaton or temporal logics [26]. Nevertheless, this formal description is not soundness and some manual translation is still needed. Model checking is used with two purposes: (i) to check if services satisfy customer's demands and and designer's specifications, and (ii) to check if orchestration satisfies liveness, safety, fairness and reachability. Different methods are used: bisimulation to verify the specification, mu-calculus to check properties as safety or reachability and pi-calculus to eliminate ill behaviors.

Nakajima claims that to verify a composite web service prior to its execution may be mandatory [51]. First, translates a WSFL description into *Promela*, the specification language for SPIN model checker. Furthermore, additional properties are expressed in LTL to be added to the model checking process. The verification process detects reachability, deadlock freedom and specific user properties.

Planning as model checking [31] is a method for solving planning problems modeling them as model checking problems. This solution is based on transition systems, but web services are a message passing paradigm, so some special considerations have to be made. Yu and Reiff-Marganiec make a formulation of the solution by modifying the strong cycle planning algorithm, which guarantees that all paths reach a solution and they are fair [80]. A four-phased algorithm is proposed. First, the planning goal and the initial knowledge are specified. After that, it automatically selects from the repository relevant web services to build the plan. In third place, the algorithm searches for plans. Finally, a physical composition step allows clients to choose the better plan, generates a executable plan specified in BPEL and monitors its execution, replanning when a failure is detected.

Walton uses model checking to validate the correctness of communication protocols between agents in an platform that integrates agents and web services [76]. The services are described in WSDL. Complex interactions among the entities that offer services are represented by the protocols, who are specified in a directly executable language called MAP. As the Nakajima's algorithm, it uses SPIN as model checker, so the specification is translated into *Promela* language. This one provides a complete automatic translation that allows non-expert to validate their services.

The main problem in all these approaches is the complexity of the state space. All of them make different simplifications to the problem to be capable of managing the validation process by limiting the number of services (or agents) and the length of the message interchanging mainly. Moreover, designer's intervention is often needed to translate service descriptions into formal languages for model checking.

Non-functional Parameters

Another point to take into consideration is to restrict the set of candidate service providers based on user-specified non-functional attributes, namely *Quality of Service* (QoS). These factors and domain specific characteristics affect on the service selection and have been taking into account in some algorithms [59][41][4]. According to Radatz, the *Quality of Service* is a set of non-functional attributes that may impact the service quality offered by a web service [36]. The main problem of this kind of information is that we cannot trust the QoS characteristics published by provider.

Different aspects of QoS might be important in different applications. Besides, different classes of web services might use different sets of non-functional attributes to specify their QoS properties. Pathak stablishes a categorization in two groups[59]: *domain-dependent* and *domain-independent* attributes. The *domain-independent* attributes represent those QoS services characteristics which are not specific to any particular service (for example, scalability or availability). On the other hand, *domain-dependent* attributes capture those QoS properties which are specific to a particular domain and most of the times are dynamic and depends on the instant in which the service is executed.

It is important to have an infrastructure which takes into account the QoS provided by the service provider and the QoS desired by the service requester so the best possible match between the two during service discovery can be found. Kokash proposes three categories to embed quality information in the service discovery process [41]:(i)solutions that rely on service providers to advertise their QoS information, (ii)solutions that rely on service clients to review service quality, (iii) solutions that rely on a third party evaluation of a web service or a provider.

Related with QoS in web services, there are some approaches that deal with the problems of relying in QoS information or with the use of this information in the matchmaking process. To deal with information reliability, service reputation is an specially interesting property that could be regarded as a measure that accumulates a user opinion about QoS in general. Kalepu et al. address this problem [37].

There are other approaches that use QoS information in the discovery process. A taxonomy for the non-functional attributes which provide a better model for capturing various domain-dependent and domain-independent QoS attributes of the services is presented by Pathak et al. [59]. These attributes allow users to dynamically select services based on their non-functional aspects. This work also introduces the notion of personalized ranking criteria, which

enhances the traditional ranking approach, primarily based on the degree of match. Furthermore, in this work a kind of ontology mapping is presented .

Following this trend, Kokash presents an approach based on the application of a distributed recommendation system to provide QoS information and on testing of retrieval methods on service specifications [41]. To improve the relevance of services to the requester in pervasive environments, Steller et al. present a service discovery model that uses semantics and context [69].

The approach presented by Aversano [4] bases the searching process on syntactic information and on service quality metrics and semantics to increase the precision of the discovery process. To take into consideration the QoS in the discovery process, the customer of the service will assign a weight to every attribute. For each selected service, a weighted sum is performed among all the attributes an the final value represents the quality of the service.

Efficient Matching

Discovery process is not a trivial problem. Due to the complexity of the underlying semantic reasoning, matching semantic web service capabilities is a heavy process. Furthermore, the matchmaking process could be intractable when the number of available services gets large. The matchmaking process should be efficient: it should not burden the requester with the excessive delays that would prevent its effectiveness.

Mockhtar describes a solution towards the efficient matching of semantic service capabilities [48]. This approach combines optimizations of the discovery process at reasoning and matching levels. Towards the optimization of the discovery process at reasoning level, they use the solution proposed by Constantinescu for encoding concept hierarchies [18]. They propose to encode classified ontologies, represented by hierarchies of concepts, using intervals. These hierarchies represent the subsumption relationships between all the concepts in the ontologies used in the directory. The main idea is that any concept is associated with an interval. Under the assumption that service advertisements and service requests already contain the codes corresponding to the concepts that they involve, semantic service reasoning reduces to a numeric comparison of codes. Furthermore, they propose to group capabilities provided by networked services into hierarchies of capabilities. These hierarchies are represented using directed acyclic graphs (DAG). If there is no matching between a requested capability and a capability situated on top of a hierarchy, its possible to infer that it will also fail with all the other capabilities contained in the sub-hierarchy of this capability in the graph. On the other hand, if a matching between a requested capability and a capability situated at the bottom of a hierarchy we can infer that the matching will also succeed with all the predecessors of this capability.

Constantinescu emphasizes the need of efficient indexes and search structures for directories. Towards this goal, they propose to numerically encoded service descriptions in OWL-S. This is done by numerically encoding ontology class and property hierarchies by intervals. More precisely, each classes (resp. property) in a classified hierarchy is associated with an interval. Then,

each service description maps to a graphical representation in the form of a set of rectangles defined by the sets of intervals representing properties combined with the set of intervals representing classes. Furthermore, for efficient service retrieval, the authors base their work on techniques for managing multidimensional data being developed in the database community. More precisely, they use the *Generalized Search Tree* (GiST) algorithm for creating and maintaining the directory of numeric services. Combining encoding and indexing techniques performing efficient service search.

Srinivasan et al. present an approach to optimize service discovery in a UDDI registry, augmented with OWL-S for the description of semantic web services [68]. This approach is based on the fact that the publishing phase is not a time critical task. Therefore, the authors propose to exploit this phase to pre-compute and store information about the incoming services. More precisely, a taxonomy that represents the subsumption relationships between all the concepts in the ontologies used by services is maintained. This proposal increases the time spent for publishing service advertisements, but it considerably reduces the time spent to answer a user request compared to approaches based on on-line reasoning.

Other Trends

Related to discovery and composition process other trends have appeared. The Torino group [25] suggests the use of reasoning techniques (inferences over a knowledge-based representation, planning, ...) to achieve adaptivity in the interaction with users and gaining flexibility in discovery process.

In [44] the proposed matchmaking approach uses fuzzy logic and user values to find a suitable service. For the ranking of service matches a match score is calculated whereby the wight values are either given by the user or estimated using a fuzzy approach. Furthermore, an evaluation of both weight assignment approaches is conducted identifying the scenarios in which one works better than the other.

The majority of the algorithms presented in the area of web services faces the discovery problem using the IO's parameters and trying to solve problems such as service composition or cross ontologies and also in some of them non-functional parameters are taken into consideration. But they do not explode all the possibilities that semantic offers and; furthermore, only a few of them take into account new interesting ideas from related areas such as multiagent systems to achieve more flexibility in that process.

2.3.3 Service Discovery based on complex discovery model and negotiation/contracting.

It comprises discovery models based on interaction protocols, forwarding QoS and privacy requirements, negotiation dialogs for refining discovery and establishing service requirements.

There are some proposals that consider the global schema of execution, which is given by the choreography, in the matchmaking process. Baldoni et al. face the problem of automatic selection and composition of web services by taking into account service descriptions with richer information [8]. They not only consider the traditional inputs, outputs, preconditions and effects properties they also consider interaction protocols. Web services are viewed as software agents, communicating by predefined sharable interaction protocols. Moreover, they use a logic specification of the interaction protocols which makes possible to apply reasoning techniques in order to introduce some degree of flexibility, that is necessary for personalizing the selection and composition of web services in an open environment. The main advantages of the proposed approach are that it leaves out services that would in no case allow the achievement of the user's goals respecting all the requirements, as well as it allows the exact identification of those permitted coursers of interaction that satisfy them. The agent will know in advance if it is worthwhile to interact with that partner.

The task of selecting a web service, that should play a role in a choreography (rather than using the choreography as the design of a new set of services), implies verifying two things: the conformance of the service to the specification of a role of interest, which guarantees that the message exchange will produce correct and accepted conversations, and that the use of that service allows the achievement of the goal, that caused its search. The achievement of the goal depends on the operation sequence because each operation can influence the executability and the outcomes of the subsequent ones. Performing a match operation by operation, does not preserve the global goal [7]. They also show how to overcome these limits by exploiting the choreography definition. Actually, it is possible to extract from the choreography some information that can be used to bias the matching process so that the global goal will be preserved.

This kind of service discovery has been used in multiagents systems [16] and in e-market environments. Agent-based services discovery mechanisms can also extends existing mechanisms by considering the types of interactions that services can use. The efficiency and precision of the matchmaking process can be enhanced by including this kind of information.

2.4 Service Discovery in Agent Based Systems

Agent orientation is an appropriate design paradigm to enforce automatic and dynamic collaborations, especially for e-business systems with complex and distributed transactions. Agent paradigm has technical advantages in software construction, legacy systems integration, transaction-oriented composition and semantics-based interaction. For this reason, many ideas from research in multiagent systems could be used in service-oriented computing approaches.

Nowadays, service oriented computing (SOC) brings additional considerations, such as the necessity of modelling autonomous and heterogeneous components in uncertain and dynamic environment. Such components must be autonomously reactive and proactive yet able to interact flexibly with other

Algorithm	Language	IO'S/ IOPE'S	Composition	Cross Ontologies	QoS
Paolucci02	DAML-S	IO's			
Abela02	DAML-S	IO's			
Constant.02	DAML-S	IO's	√		
Cardoso02	DAML-S	IO's		√	√
Lei03	DAML-S	IO's			
Wolf03		IO's			Soft constr.
Bansal03	DAML-S	IO's	√		
Brogi03	OWL-S	IO's	√		
Benata.03	DAML-S	IO's	√		
Aversa.04	DAML-S	IO's PE's?	√	√	√
Sriniv.04	OWL-S	IO's		√	
KluS05	OWL-S	IO's PE's?	√		
Pathak05	OWL-S	IOPE's		√	√
Hashem.05	OWL-S	IO's	√		
Yang05	OWL-S	IO's	√		
Kokash05	WSDL	IO's			√
Klusch06	OWL-S	IO's		√	
Brogi06	OWL-S	IO's	√	√	
Mokhtar06	DAML-S	IO's	√		
Corfini06	OWL-S	IO's	√	√	
Bailey06		Functionalities			√
Prabhu07		IO's	√		
Baldoni07		IOPE's - Roles	√		

Table 2.1: Service Discovery Algorithms

components and environments. As a result, they are best thought of as agents who collectively form MAS. SOC represents an emerging class of approaches with MAS-like characteristics for developing systems in large-scale open environments. Key MAS concepts are reflected directly in SOC with ontologies, process models, choreography, directories and facilitators, service level agreements and quality of service measures. In this section, the ideas and agents features used in the area of service discovery and how matchmaking algorithms have used them to improve the service discovery process are presented. Classification of the matchmaking algorithms taking into account if they are used in centralized or distributed environments is also given 2.4.2.

2.4.1 Agent features in matchmaking algorithms

In the scope of matchmaking, agent paradigm, due to its features can be taken into consideration to achieve more flexibility and better results in service discovery algorithms. In this section some agent features used in service discovery

are presented.

Interactions and conversations

In multiagent systems, agents replace the procedure call method for service invocation by a communication-based mechanism. This allows complex interactions instead of simple input-output schemas.

This agent feature allows us to provide another way to discover and invoke (request) services in a more flexible and suitable way. There are some proposals for web services that consider the global schema of execution in the matchmaking process, which is given by the choreography. Baldoni et al. face the problem of automatic selection and composition of web services taking into account interaction protocols [8][7]. Web services are viewed as software agents, communicating by predefined sharable interaction protocols.

Furthermore, there are some approaches as WSIG[32], WSDL2JADE[52] or WSAI[2] that allows requests using FIPA messages and interaction protocols to demand services provided by agents or by web services.

Recommendation

Web services have an implicit knowledge about the world, while agents are expected to base their knowledge and their actions in the observable behavior of other agents. When an entity, agent or service, request a service, the entity is being confident with the provider because it depends on it for having the work done. Reputation and Trust are measures for maintaining this confidence and select trustful services. Some approaches in the agent area propose a framework for semantic discovery and selection of web services using a trust management model that consist on capturing user's trust disposition, verifying trustworthiness level, making trust decision and evaluation after a transaction [65].

There are other approaches that use a knowledge called community culture to facilitates the discovery of web services satisfying user needs. Community culture is based in the knowledge about acting effectively in the environment, which is often implicit and specific to the community. Kokash in [41] presents a system based on the implicit culture that uses the history of user-systems interactions and client-service communication logs to provide recommendations on web services.

Many approaches consider ratings of service providers based on subjective opinions of web service users. For instance, Manikrao describes a service selection framework which combines a recommendation system with semantic matching of service requirements [45]. There are other proposals where agents act as proxies to collect information and to build the reputation of semantic web services [46].

In other approaches, the objective experience data of agents is used in order to evaluate its expectation from a service provider and to make decisions using its own criteria and mental state [64]. Basically, service consumers collect previous experiences form other service consumers with similar requirements and

make decisions using different methods. By simply sharing their experiences, service consumers lead to the emergence of a consumer society in which the overall satisfaction increases.

An extension of the algorithm used in IMPACT (platform for collaborating agents)[70] is presented by Zhang[81]. This extension consist in adding to the usual service description (service name, inputs, outputs and attributes) an extra component that consists of a registry that keeps the evaluation that other consumers gave to the service. The registry is formed by pairs of two elements: the number of times that it has been delegated and the satisfaction degree of the executed service.

Cooperation and roles

Agents are cooperative, forming teams and coalitions that provide higher-level services. In these coalitions, agents can play several roles depending on the situation or their interests. This feature can be exploited in agent-based service discovery mechanisms, where the information provided by the organizational model underlying the multiagent system can be used. In order to improve the efficiency and the usability of agent-based service-oriented architectures, Cáceres et al. suggest exploiting common organizational concepts such as social roles and types of interactions to further characterize the context that certain semantic services [16] [23]. Following this idea, a matchmaker which makes use of roles and interactions types is presented. The matchmaker is an extension of the hybrid matchmaker OWLS-MX[39] that improves the efficiency and precision of the matching process.

Negotiation

Web services, as currently are defined, are not considered totally autonomous. Autonomy among agents is understood as a social autonomy, where agents are aware of the existence of other agents and they are sociable and can cooperate to achieve common goals. This agent feature allows negotiations where the agents can reach agreements in service composition, or in non-functional parameters related with services. There are some approaches in the discovery process that use negotiation techniques and protocols to select the suitable services.

Negotiation protocols are another mechanism used in multiagent systems in e-commerce or online supply chains applications. Participant agents negotiate about the properties of the services they request and provide to enter into binding agreements and contracts with each other. Dang presents in [20] a protocol that supports many-to-many negotiation in which many agents negotiate with many other agents simultaneously using colored petri nets.

Bircher proposes to implement an environment based on agents and marketplaces where agents, representing wireless service customers, can detect and meet other agents (representing wireless network service providers) negotiate with them about the offered services and reserve the resources for the agreed price [12].

In electronic commerce, negotiation is a common method in the discovery tasks. Ouksel presents an example where the organization needs a matchmaking process in which provider agents are matched with demanding agents [55]. These matched agents can establish negotiations directly or with the control of a referee entity. The matchmaking process allows to match the agents with the less conflict interests. The algorithm matches each provider with a buyer, the problem can be seen as a not directed biparted graph, complete and weighted. The aim is to find the suitable matching that maximizes the sum of the weighed edges.

2.4.2 Matchmaking algorithms in centralized and distributed environments

In open multi-agent systems, agents can be asked for tasks that they alone can not achieve. But agents can delegate these tasks to other agents or cooperate with them. So agents need to know which services can be provided by other agents and how to contact with them. The solutions to the service discovery problem in multi-agent systems follow two lines: a centralized approach and a distributed one. In this section we review both lines with more detail.

Service discovery algorithms in centralized middle agents

Regards to matchmaking algorithms, agents and web services have some aspects in common. This is more obvious in the algorithms used by middle-agents. The approach based in middle agents is used in open and dynamic systems where the scalability and the workload are low. The main advantage is that matchmakers could provide an optimal matching because they can take into account all the registered services in the system. Furthermore, middle-agents usually make an efficient search and get a good throughput. The most important drawback is that this kind of agents could be a bottleneck in systems with high work load. They are also complex, they need a huge amount of memory to keep advertisements, and they should understand different languages.

One of the firsts matchmaking algorithm used in agents systems was presented by Sycara et al.[71]. It uses its own language, LARKS. It is an expressive language able to support automatic inferences. The implementation of the matchmaking process for LARKS specifications uses different techniques from information retrieval, IA and software engineering that calculate semantic and syntactic similarity between the advertisements and requested descriptions of the agent capabilities. Matchmaking engine consists of five filters that progressively restricts the number of advertisements that are candidates for a match: the higher the precision is, the longer the time the matchmaker needs before delivering an answer. These filters can be configured by the user to reach the desired solution. The disadvantage of this algorithm is that uses an own language for service description. Besides, in the matchmaking process does not consider criteria related to the user preferences or QoS. In addition, as in other

matchmaking algorithms seen for web services[58][3][77], service composition is not considered either.

Other algorithms have been developed taking into account criteria related to QoS or user preferences. Mecar[47] presents a matchmaking algorithm based on semantic information of service descriptions coded with DAML-S. The algorithm is very similar to the presented by Paolucci[58], but it considers the user preferences related to QoS. The main feature of the algorithm is that it divides the matchmaking procedure in several parts: input matching, output matching, profile matching and user matching. Each part is independent of the other three which makes the process more flexible. The final result will be based on the results of each stage of matchmaking.

Corradini et al.[19] present a matchmaker agent which its aim is to obtain a service with a certain QoS. With the purpose of assuring the choice of the quality of a requested service, the matchmaker communicates with a QoS authority. Besides the functions of the matchmaker agent, we could also emphasize another functions as service coordination based on protocols and the capability to obtain services considering quality and confidence.

Another important aspect for matchmaking algorithms is to be able to consider service composition. Trazec et al.[72], as in the algorithms presented previously in web services[15][13][10], present a matchmaking algorithm that uses the Process Model. The Process Model is based on the process as a key concept, and that describe the service not only in terms of inputs, outputs, preconditions and effects. When it is appropriate, the algorithm uses the service decomposition in subprocesses. The matchmaking process requires two different tasks: to abstract the required query capabilities and to compare those capabilities with the service providers available.

Flexibility in another important characteristic in matchmaking algorithms. PHOSPHOROUS[29] uses an specific language to express agent capabilities and requests. Both are automatically translated in descriptions which are organized in a subsumption hierarchy that exploits the descriptions in the ontology domain. PHOSPHOROUS reasons with the complete expression that specifies capabilities and requests, including parameters and whatever constraint in the capability constraints. PHOSPHOROUS uses a matching based on the reverse of subsumption to find agents whose capabilities are subsumed by the capabilities of the request. Therefore, they can satisfy some aspects from the original request. In other cases, it could be possible to complete the request splitting it and expressing it in a different terms. PHOSPHOROUS also allows query reformulation, providing a more flexible service.

To improve the service matchmaking, algorithms could use information related with the system in which the agents are. Cáceres et al. present an approach that takes into account aspects as organizations, roles or interactions [16]. The mechanism for service discovery in multiagent service uses the information provided by the organizational model of the system. Following this idea, a matchmaker which make use of roles and interactions types is presented. The matchmaker is an extension of the hybrid matchmaker OWLS-MX[39] that improves the efficiency and precision of the matching process. The main rou-

tine of the algorithm calculates the matching degree between the requester role and a role of a service advertisement. For each role, the matching with the service advertisement and the matching between the necessary and requester roles, are calculated. The matching degree is the minimum difference between both values. The semantic match between two roles is calculated depending on the role ontology. It depends on two values: the match degree and the distance between roles in the ontology. Another extension of OWLS-MX is presented in the CASCOM abstract architecture[23], which combines agent technology and semantic web services, peer-to-peer, and the mobile computation for service mobile environments. This architecture presents, in its service coordination layer, a semantic discovery service that is composed of two types of agents: Service Discovery Agents (SDA) and Service Matchmaking Agents (SMA).

Service discovery algorithms in distributed entities

Different approaches are suggested to overcome the above mentioned problems related with the centralized paradigm. These solutions distribute the information about agent capabilities among other elements of the system. These approaches are suggested in environments with high scalability and workload. The main advantage that they provide is that they have not only one fail point. Furthermore, they provide a decrease in time communications and spread the memory needed by agents. The main drawback is that some distributed approaches, such as coalitions or peer-to-peer, do not guarantee a matching or the best matching and, in some situations, the broadcast could overload system communications.

Peer-to-peer. Peer-to-peer approach takes advantage of the fact that each agent already knows its own capabilities and those of a few peers, and uses peer-to-peer search (recursively) for locating agents with the needed capability [55]. An agent broadcasts a query to its neighbors local knowledge and the agent that receives such a request either offers its services to the original caller or broadcasts the request to its own neighbors. This approach relies on each agent indeed holding a neighbor list of its peers and adhering to the location protocol. The drawback of this approach to service discovery is that the communication among agents in is essential and the overall communication traffic overhead may be large. It also requires a connection model among the agents that will ensure a high number of correct answers (good hit ratio for queries) and in the same time control communication overhead so that the network is not overwhelmed by the messages of the location protocol. This approach is used, as we have seen in the previous section, in electronic commerce[55].

Negotiation protocols, as we have seen previously, are another mechanism used in multiagent systems in e-commerce or online supply chains applications. Participant agents negotiate about the properties of the services they request and provide to enter into binding agreements and contracts with each other [20] [12].

Coalitions. Another way to locate distributed services is to form coalitions o clusters. Nevertheless, the choice of what coalitions are going to be formed is a difficult task. This entails recursively to calculate the values of the coalitions and later selecting the coalition with the best result. The calculation of the coalition values can be made in parallel, but this phase requires that each agent knows the rest of agents of the system (global knowledge). In addition to determine the best value, they have to use broadcast. Therefore, in some situations, the system could be overload.

Ogston and Vassiliadis present an algorithm for consumer agents that only wants a suitable matching but not the best [54][53]. In this algorithm, each agent has a number of tasks and needs to delegate in other agents. This agents are located randomly around its neighbors, checking if its characteristics are similar to the searched features. When a matching between two agents is founded, it is considered that these agents are able to cooperate. These agents get into a coalition which allows each agent to extend its neighbors and the scope of search is extended for future tasks. In the case that the agent do not find another agent with the desired characteristics, the agent will look for them in other coalitions.

The discovery mechanism presented by Moore and Sura [49] is based on the use of relations. With the similarity of keywords, historical information and clustering, each agent can determine which relations should choose. Similarity among keywords is the ratio of keywords that an agent and its partner of relation have in common. Agents with similar keywords will be near in the relationship network. Furthermore, inside the clusters, it could be possible to create new subclusters in which the second keyword is shared among the members. In this algorithm, the historical relations are presented as an improvement of the algorithm.

Distributed middle-agents. Another way for agents to locate services in a more efficient way is the distribution of the middle agents o facilitators[50]. This approach consists on the distribution of the service directory, its memory and the pass messaging cost. Jha et al. propose to split the function of the facilitator among a group of agents [35]. The system designer assigns a local matchmaker to each host or segment of the system, which provides matchmaking services to agents in its vicinity (its segment). The local matchmaker can consult its peers or a central matchmaker whenever it cannot provide an answer to a local query. This type of solution reduces communication traffic and confines it to network segments (in which communication is fast). Moreover, it reduces message queue sizes, improving scalability and fault tolerance. This approach is applicable in systems that have a hierarchical topology, in which information sharing can be confined to local segments.

In systems with very large segments the problems of scalability are only marginally relieved by this approach (because the large segments become overloaded systems which have local bottlenecks). Another case in which this approach is not useful is in systems with many cross-links between segments. In this case the overhead of coordinating tasks among local matchmakers might be

Approaches	Problems	Suitable Environ.	Advantages	Drawbacks
Centralized:	less fail tolerance bottleneck	low workload low scalability	efficient search good throughput optimal match	complex memory manage great quantity of queries
Distributed:	comm. overload coord. overload	high scalability high workload	direct queries spread information better fault tolerance	couldn't find match low throughput if no interactions between neighbors

Table 2.2: Service discovery in agent based systems

greater than the benefit obtained from their distribution.

Sigdel et al. present an adaptative system [66]. The framework suggested allows automatically adaptable matchmaking methods for service localization depending on the network structure and characteristics. This approach is based on two levels: system adaptation level and node adaptation level.

In the system adaptation level the system adapts itself to the changing circumstances of the network, the number of nodes and the service load. If anyone of these circumstances increases, the system introduces new matchmakers that will reduce the service load of the central matchmaker. These new matchmakers are defined in a segment of consumers and suppliers where they could be created. When some of the previous circumstances decrease, for example the service load, a mechanism unify the segments and eliminates the created matchmakers to increase the productivity of original matchmaker.

In the node adaptation level, nodes suppliers or consumers could be promoted to matchmakers with small modifications. When matchmakers are not required in the system they could return to consumers or suppliers. In each segment, there is a matchmaker in charge of looking for the matchmaking between consumers and suppliers. If the matchmaker cannot find a suitable matching it sends the request to others matchmakers. The communication and cooperation between matchmakers are fundamental.

2.5 Final Remarks

In sections 2.3 and 2.4 we have described and showed examples of works related to service discovery and composition. In this section we elaborate on important topics that emerged when examining these works. While the main sections of this chapter aims to act as a road map of service discovery, herein we critically review issues concerned with the information used in the discovery process, if compositions is considered or if crossing ontologies have been taken into account.

We start by discussing the proposals in the scope of web services. Most of them, basically based on IO's. Paolucci's algorithm is limited to discover simple services. It does not consider the composition discovery nor the use

Algorithms	Centralized Env.	Distributed Env.	Main Features
LARKS82	√		Own Language
Mullen88		Distr. Middle-Ag	distributed directory
Corradini88	√		QoS Use of protocols
Jha98		Distr. Middle-Ag	Comunication segments Hierarchical topology
Phosphor.01	√		reverse of subsumption request splitting reformulation
Ogston01		Coalitions	Clustering
IMPACT01	√		Trust and reputation
Moore02		Coalitions	Clustering Similarity keywords Historical relations
Trzec04	√		Service composition
Ouksel04		P2P	Negotiation
Bircher04		P2P	Negotiation
Mecar05	√		User preferences Flexible
Sidgel05		Distr. Middle-Ag	Adaptative system
Cáceres06	√		Roles
Dang06		P2P	Negotiation

Table 2.3: Agents service discovery in distributed and centralized environments

of different ontologies. Another lack is that the process of matching does not consider parameters related with quality of service (QoS); it is only based on the input and output parameters. From this algorithm arose others that made some modification to deal with some of its deficiencies [3][39][4][17] [43] [18].

There are some algorithms that, apart from IO's information, take into account preconditions and postconditions (IOPE parameters). This information can be important to be taken into consideration in negotiations and allow making inferences about that. In some situations, for example, if a user is looking for a rent car service with a normal credit card and there are two services s_1 and s_2 that offer that service. s_1 only allows to pay with special kind of credit card and s_1 accepts any kind of credit card. The discovery process returns as a solution both services but in fact, the user are only interested in s_2. For these reason, it could be interesting to take into account preconditions and postconditions to restrict the possible suitable services.

Where the conventional IOPE-based matchmaking algorithms are unable to determine suitable matches for a request, is decisive take into account compositions. This characteristic is present in some discovery algorithms [4][10][15] [38][11][48][34][18]that achieve more flexible matching. Most of them use the information provided by the service model to achieve this aim. Although these kind of algorithms have a worst case timing analysis, the average case performance is actually much better since most of the possible advertisement process

nodes are not explicitly examined.

In some of the presented algorithms important points as crossing ontologies [17][4][59][13][39] or Quality of Service (QoS) to restrict the set of candidate service providers based on user-specified non-functional attributes [59][41][4] are taken into account.

Although the features that are present in the algorithms that improve service discovery, the algorithms also have some weakness. For example, in most of them they do not exploit all the data provided in the service profile. The majority of the algorithms use IO's but forget the preconditions and postconditions and the capability to make inference about this data to obtain new information that could be useful to eliminate false suitable services. Furthermore, they not emphasize in other aspects that could give them more flexibility in the discovery process such as allowing partial matching or fuzzy data. There are other points which limit the capability of new ways of matchmaking. For instance, web services nor have internal state and neither awareness of changes in its environment. This fact makes a web service not be able to take advantage of new capabilities in their environment or customize their services to users such as providing improved services. Another point is synchronous communication. As a consequence of that, hardly ever you can see algorithms that use complex interactions similar to FIPA protocols in agents. In addition, web service discovery algorithms do not consider temporal constraints over services.

Operations as service discovery can also be realized through agent-oriented techniques. This paradigm is appropriate to automatic and dynamic collaboration especially for systems with complex and distributed transactions. Agents present an extended proposals that provide a more flexible and efficient matchmaking process. Some of the proposals use algorithms very similar to web services but they add agent features to achieve a more efficient and flexible matching. We have seen in this chapter that the use of roles[16] [23], interaction patterns[8][7], trust [65][41][45][46] or negotiation [20][12][55] could be another suitable way of afford discovery problem. Furthermore, agents presents some features such as awareness of theirs internal states or asynchronous message interchange which allows to establish conversations and negotiations which provides more flexibility. Similar to web service discovery approaches, in agent-based approaches are some open issues to consider such as QoS in the matchmaking process and also take into account temporal constraints. These characteristics are useful to limit the possible candidates to be a suitable provider and to make easier to find the requested service. In some agent and web services proposals, these ideas are being taken into account, but it is difficult task due to the complexity of the required solution.

Chapter 3

THOMAS Architecture

3.1 Introduction

The areas of Service Oriented Computing (SOC) and Multi-agent Systems (MAS) are getting closer and closer. Both trying to deal with the same kind of environments formed by loose-coupled, flexible, persistent and distributed tasks. Moreover, MAS must cooperate with others inside a "society". Due to the technological advances of recent years, the term "society", in which the multi-agent system participates, needs to meet several requirements such as: distribution, constant evolution, flexibility to allow members enter or exit the society, appropriate management of the organizational structure that defines the society, multi-device agent execution including devices with limited resources, and so on. All these requirements define a set of features that can be addressed through the open system paradigm and virtual organizations in multiagent systems. There are other problem that should be addressed: the integration of multiagent system paradigm and the service-oriented computing paradigm. Nowadays, service oriented computing (SOC) brings additional considerations, such as the necessity of modelling autonomous and heterogeneous components in uncertain and dynamic environment. Such components must be autonomously reactive and proactive yet able to interact flexibly with other components and environments. As a result, they are best thought of as agents who collectively form MAS. SOC represents an emerging class of approaches with MAS-like characteristics for developing systems in large-scale open environments. They key MAS concepts are reflected directly in SOC with ontologies, process models, choreography, directories and facilitators, service level agreements and quality of service measures. For these reasons it is also interesting to integrate these two technologies to model autonomous and heterogeneous computational entities in dynamic and open environment.

In this chapter THOMAS (MeTHods, Techniques and Tools for Open Multi-Agent Systems) is presented as an architecture that deals with the integration of agents and services, being agents complex entities that can handle the prob-

lem of service discovering and composition in dynamic and changing open environments [63][33]. These agents are organized not in plain societies, but in structured organizations that enclose the real world with the society representation and ease the development of open and heterogeneous systems. Current agent platforms must integrate these concepts to allow designers employ higher abstractions for modeling and implementing these complex systems. All these concerns are gathered in the THOMAS architecture.

3.2 Architecture Model

THOMAS architecture basically consists of a set of modular services. Though THOMAS feeds initially on the FIPA architecture, it expands its capabilities to deal with organizations, and to boost up its services abilities. In this way, a new module in charge of managing organizations has been introduced into the architecture, along with a redefinition of the FIPA Directory Facilitator that is able to deal with services in a more elaborated way, following Service Oriented Architectures guidelines. As it has been stated before, services are very important in this architecture. In fact, agents have access to the THOMAS infrastructure through a range of services included on different modules or components. The main components of THOMAS are the following (Figure 3.1):

- *Service Facilitator* (SF), this component offers simple and complex services to the active agents and organizations. Basically, its functionality is like a yellow page service and a service descriptor in charge of providing a green page service.
- *Organization Management System* (OMS), it is mainly responsible of the management of the organizations and their entities. Thus, it allows creation and management of any organization.
- *Platform Kernel* (PK), it maintains basic management services for an agent platform.

3.3 SF: Service Facilitator

The *Service Facilitator (SF)* is a service provider which is in charge of service management (service discovery and composition) and also of the access to the THOMAS platform. The SF offers all services needed for a suitable service management and access performance. These services have also to be used by the rest of THOMAS components (OMS and PK) to advertise their own services. Services that are going to be registered in the SF should be described in a OWL-S specification for semantic web services, extended when needed to empower its functionality. The service description should be a tuple:

```
<serviceID, goal, profile, process, grounding, ontology>
```

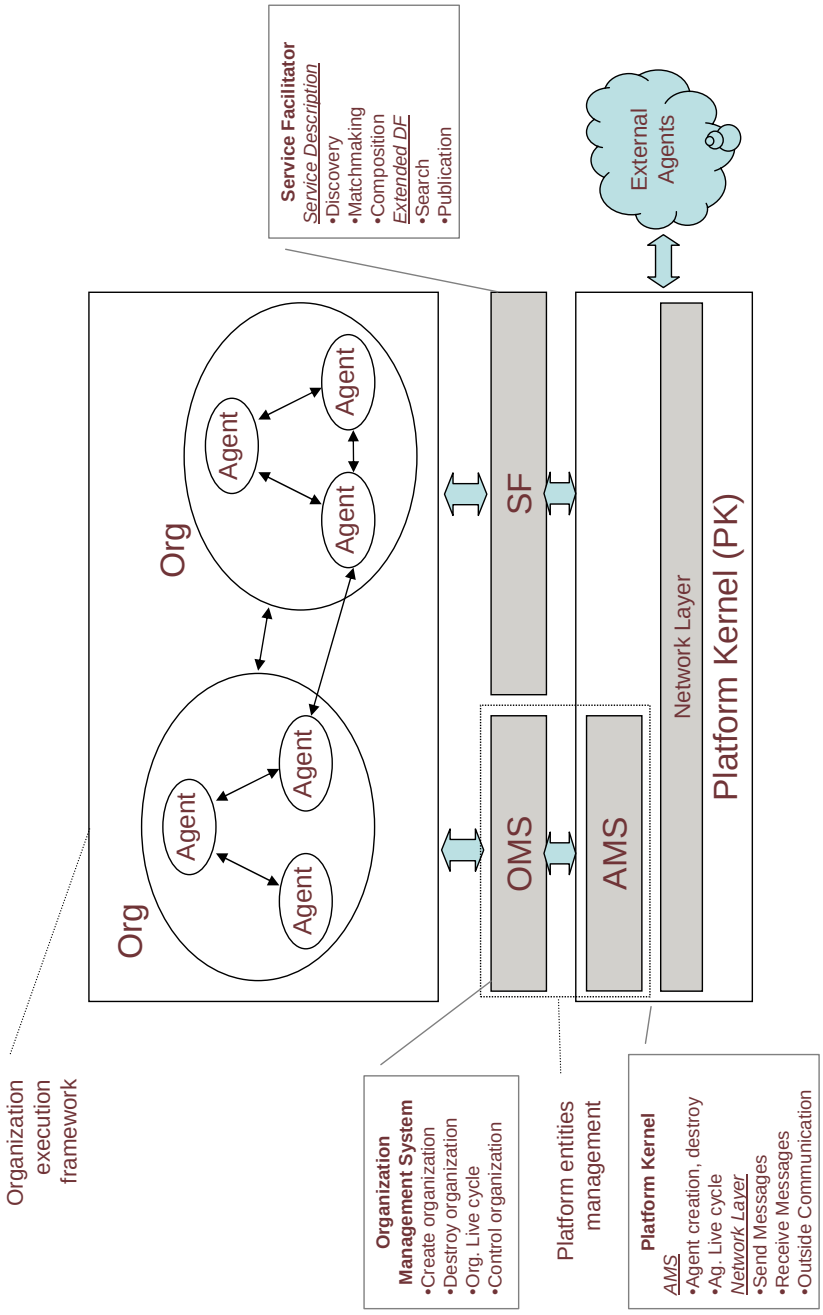

Figure 3.1: THOMAS components

and it is organized in two parts: one related with general information of the service and the other related with a low-level.

To manage services the SF offers a set of meta-services that can be classified in three types: registration, affordability, discovery. It is briefly described as follows (Table 3.1). In the next chapter the SF is described with more details.

Type	**Meta-service**	**Description**
Registration	RegisterProfile	Creates a new service description (profile)
	RegisterProcess	Creates a particular implementation (process) for a service
	ModifyProfile	Modifies an existing service profile
	ModifyProcess	Modifies an existing service process
	DeregisterProfile	Removes a service description
Affordab.	AddProvider	Adds a new provider to an existing service process
	RemoveProvider	Removes a provider from a service process
Discovery	SearchService	Searches a service (or a composition of services) that satisfies the user requirements
	GetProfile	Gets the description (profile) of an specific a service
	GetProcess	Gets the implementation (process) of an specific a service

Table 3.1: SF meta-services

3.4 OMS: Organization Management System

The *Organization Management System (OMS)* is in charge of organizations life-cycle management, including specification and administration of both their structural components (roles, units and norms) and their execution components (participant agents and roles they play and active organizational units).

Organizations are structured by means of *organizational units*, which represent groups of entities (agents or other units), that are related in order to pursue a common goal. Those organizational units have an internal topology (i.e. hierarchical, team, plain), which imposes restrictions on agent relationships and control (ex. supervision or information relationships).

In THOMAS, a "virtual" unit has been defined in order to represent the "world" system in which agents participate by default. The OMS creates organizations inside this "virtual" unit, by means of registering organizational units, which can also be composed of more units. Moreover, roles are defined in each unit. They represent all required functionality needed in order to achieve the

Type	Subtype	Meta-service	Description
Structural	Registration	RegisterRole	Creates a new role inside a unit
		RegisterNorm	Includes a new norm inside a unit
		RegisterUnit	Creates a new unit inside a specific organization
		DeregisterRole	Removes a specific role description from a unit
		DeregisterNorm	Removes a specific norm description
		DeregisterUnit	Removes a unit from an organization
	Information	InformAgentRole	Indicates roles adopted by an agent
		InformMembers	Indicates entities that are members of a specific unit
		QuantityMembers	Provides the number of current members of a specific unit
		InformUnit	Provides unit description
		InformUnitRoles	Indicates which are the roles defined inside a specific unit
		InformRoleProfiles	Indicates all profiles associated to a specific role
		InformRoleNorms	Provides all norms addressed to a specific role
Dynamic	Basic	RegisterAgentRole	Creates a new <*entity, unit, role*> relationship
		DeregisterAgentRole	Removes a specific <*entity, unit, role*> relation
	Compound	AcquireRole	Requests adopting a specific role inside a unit
		LeaveRole	Requests leaving a role
		Expulse	Forces an agent to leave a specific role

Table 3.2: OMS meta-services

unit goal. They might also have associated norms for controlling role actions (i.e. which services agents playing that role are allowed to request, offer or serve; permissions for accessing resources). As a result, agents can dynamically adopt roles inside units, so the OMS controls this role adoption process and which are

the entities that play each role through time.

The OMS component makes use of the following information:

- *UnitList*: it stores existing units, together with their objectives, topology and parent unit.
- *RoleList*: is stores the list of roles defined in each unit and their attributes (accessibility, visibility, position and inheritance). *Accessibility* indicates whether a role can be adopted by an agent on demand; *Visibility* indicates whether agents can obtain information of this role on demand; *Position* indicates whether it is a supervisor, subordinate or member of the unit; and *Inheritance* indicates its parent role.
- *NormList*: it stores norms defined in the system.
- *EntityPlayList*: it describes ¡*entity, unit, role*¿ association, i.e. which roles have been adopted by an entity (agent) inside each unit.

The OMS offers all services needed for a suitable organization performance. These services are classified as: *structural services*, that modify the structural and normative organization specification; and *dynamical services*, that allow agents to entry or leave the organization dynamically, as well as role adoption. The complete list of the OMS services is detailed in Table 3.2. Those services are briefly described as follows.

3.5 PK: Platform Kernel

The Platform Kernel (PK) is in charge of providing the usual services required in a multi-agent platform. Therefore, it is responsible for managing the life cycle of the agents included in the different organizations, and also allows to have a communication channel (incorporating several message transport mechanisms) to facilitate the interaction among entities. On the other hand, the PK offers a safe connectivity and the necessary mechanisms that allow multi-device interconnectivity.

A previous security mechanism is assumed for some of the services described below, which permits to manage who can invoke each service and over whom. For example, the supervisor of an organization may have the option of creating new agents inside its organization. For this, the agent *Register* Service should be invoked at platform kernel level.

The services offered must be FIPA legacy, with some modifications. The PK services needed in a THOMAS infrastructure are classified in four types: (i) *Registration*: they allow to add, modify and remove native agents from the platform; (ii) *Discovery*: services to get some information about the native agents active in the platform; (iii) *Management*: services to control the activation state of native agents in the platform; (iv) *Communication*: services to communicate agents in the platform and outside it.

The complete relation of the PK services is detailed in Table 3.3. Those services are briefly described as follows.

Type	Service	Description
Registration	Register	Registers a new agent in the platform
	Deregister	Eliminates an agent registration
	Update register	Modifiies the information appearing in an agent register (except the agent name).
Discovery	Agent Search	Request information from a registered agent on the platform.
	Get Description	Obtain the platform description.
Management	Suspend	Suspend the execution of an specific agent.
	Activation	Activate the execution of an agent who currently is suspended.
Communication	Send	Send a message to any agent in the platform or outside it.

Table 3.3: PK services

Chapter 4

Service Facilitator

4.1 Introduction

The Service Facilitator (SF) is a mechanism and support by which organizations and agents can offer and discover services. The SF provides a place in which the autonomous entities can register service descriptions as directory entries.The SF deals with one of the THOMAS objectives: develop intelligent service coordination techniques/methods within open, decentralized multiagent systems: intelligent service location (directory services, syntactic and semantic comparison techniques for services) and generation and adaptation of composed services.

To deal with the objective of service composition AI planning techniques can be employed to automate the process. PDDL planners are widely used in service composition. The reasons of this fact are that PDDL is widely recognized as a standardized input for state-of-the-art planners and OWL-S has been strongly influenced by PDDL language, mapping from one representation to another is straightforward (as long as only declarative information is considered). When planning for service composition is needed, OWL-S descriptions could be translated to PDDL format [62].

Another important point in service composition is the service descriptions and the information that is used in that process. In many cases only the inputs and outputs are considered and non-functional information is only used for ranking the service. An important non-functional parameter to take into account is the time. In many cases service duration is assumed to be a snapshot and it is not taken into account in service composition. But time parameter is important to consider. Sometimes service quality decays with time and, in some way, the provider have to determine how much time takes the service to be executed. If the execution of the service takes too much time, maybe the service has not interest for the user. In service compositions it is also important to consider the time to select the less time consuming composition if the user is concerned about that feature.

One of the objectives of the SF is provide an intelligent service location (directory services, syntactic and semantic comparison techniques for services) and generation and adaptation of composed services. To deal with this objective and taking into account the situation presented in the previous paragraphs a proposal is presented. This proposal consists of an OWL-S extension to include temporal qualified services by including duration as a non-functional parameter and temporal constraints in the preconditions and effects of the service. The temporal annotated services in OWL-S are translated in durative actions in PDDL 2.1, so any planner that deals with that language can be used.

4.2 Service Facilitator: High-Level Design

The Service Facilitator (SF) is a mechanism and support by which organizations and agents can offer and discover services. The SF acts as a:

- *Gateway* to access the THOMAS platform. It manages this access transparently, by means of security techniques and access rights management.
- *Discovery and Composition Service* which searches a service (or service composition) for a given service profile or goals that can be fulfilled when executing the service (or service composition). This is done using the matchmaking and service composition mechanisms that are provided by the SF.
- *Yellow Pages Manager* which can find which entities provide a given service.

A service represents an interaction of two entities, which are modeled as communications among independent processes. Regarding service description, a service offers some capabilities, each of which enables fulfilling a given goal. The service may have some preconditions, which need to be true for the service execution. Moreover, both service client and provider exchange one or more input and output messages during the service execution, which has some effects on their environment. Furthermore, there could be additional parameters in a service description, which are independent of the service functionality (non-functional parameters), such as quality of service, deadlines and security protocols. Finally,the service results can be enhanced using automatic service composition mechanisms (for example, partial matchmaking). To do this the SF maintains the description of the internal processes that are executed when the service is running.

In our case, the Multi-agent Technology provides us with FIPA communication protocols which are well established mechanisms in order to standardize the interactions. In this way, every service has an associated protocol. In those cases in which the service requires the execution of a chain of protocols, the service is marked as complex. Taking into account that THOMAS works with semantic services, another important data is the ontology used in the service. Thus, when the service description is accessed, any entity will have all needed

information in order to interact with the service and make an application that can use this service.

Normally, a service can be supplied by more than one provider in the system. In this way, a service has an associated list of providers. All providers can offer exact copies of the service, that is, they share a common implementation of the service. Or they may share only the interface and each provider may implement the service in a different way. This is easily achieved in THOMAS because the general service profile is separated from the service process.

A service is defined as a tuple $(sID, goal, prof, proc, ground, ont)$:

- *sID* is an unique service identifier;
- *goal* is the final purpose of the service and it provides a first abstraction level for service composition;
- *prof* is the service profile that describes the service in terms of its IOPEs (Inputs, Outputs, Preconditions and Effects) and non-functional attributes, in a readable way for those agents that are searching information (or matchmaking agents which act as searching service agents). This type of representation includes a description of what the service fulfills, the constraints about its applicability and the quality of service, and the requirements that clients have to satisfy in order to use the service.
- *proc* describes how a client has to use the service, specifies the semantic content for using the service, situations in which it is obtained, and, whenever it is required, the step by step processes to get these results. In other words, it specifies how to call a service and what happens when the service is executed.
- *ground* specifies in detail how an agent can access the service. A grounding specifies a communication protocol, the message formats, the contact port and other specific details of the service. It is specified using the OWL-S standard extended with FIPA protocols.
- *ont* is the ontology that gives meaning to all the elements of the service. OWL-DL is the chosen language.

This proposal is based on OWL-S specification for semantic web services, extended when needed to empower its functionality. Goals, preconditions and effects (or postconditions) are logical formulas.

The tuple defined above for service specification is implemented in two parts: the abstract service, general for all providers; and the concrete service, with the implementation details. In this way, services are stored inside the system split into these two parts: the service profile (that represents the abstract service specification) and a set of service processes specifications (that detail the concrete service). Thus, in THOMAS services are implemented as the following tuple, in which its elements are OWL-S extended specifications:

```
<ServiceID, Providers, ServGoal, ServProfile>
Providers ::= <ProvIDList, ServImpID, ServProcess, ServGround>+
ProvIDList ::= ProviderID+
```

where:

- *Providers* is a set of tuples composed of a Providers identifier list (ProvIDList), the service process model specification (ServProcess), and its particular instantiation (ServGround).
- *ProvIDList* maintains a list of service provider identifiers

The SF supplies a set of standard services (meta-services) to manage the services provided by organizations or individual agents. These meta-services have also to be used by the rest of THOMAS components (OMS and PK) to advertise their own services. SF meta-services can be classified in three types:

- Registration: they allow to add, modify and remove services from the SF directory. Avaliable services are:RegisterProfile, RegisterProcess, ModifyProfile and ModifyProcess.
- Affordability: for managing the association between providers and their services. Avaliable services are: AddProvider and RemoveProvider.
- Discovery: for searching and composing services as an answer to user requirements. Avaliable services are: SearchService, GetProfile and GetProcess.

Next, SF services are described with more detail.

Service 1 RegisterProfile

Precond.: $\nexists S \in SF | S.ServProfile = ServProfile$
Input: the service goal and the service profile
Output: a service ID
Effects: $\exists S \in SF | S.ServiceID = ServiceID \wedge S.ServProfile = ServProfile$

RegisterProfile: it is used when an autonomous entity (an organization or an agent) wants to register a new service description. To do this, the profile structure has to be completed in order to provide the service description.

Service 2 RegisterProcess

Precond.: $\exists S \in SF | S.ServiceID = ServiceID \wedge (\nexists I \in S.Providers | I.ServProcess = ServProcess \wedge I.ServGround = ServGround)$
Input: service ID, its process, its grounding and provider ID
Output: a unique service ID for this process (*ServImpID*)
Effects: $\exists S \in SF | S.ServiceID = ServiceID \wedge (\exists I \in S.Providers | I.ServImpID = ServImpID \wedge ProviderID \in I.ProvIDList \wedge I.ServProcess = ServProcess \wedge I.ServGround = ServGround)$

RegisterProcess: it is used when an agent wants to register a particular implementation of a given service. The ID of the service and the provider entity (*EntityID*) have to be specified. There could be several providers for the same service process. In this case, the first time an implementation is going to be added, the *RegisterProcess* meta-service has to be used. If other providers offer the same process model for this service, they can be attached to it by using the *AddProvider* meta-service.

Service 3 ModifyProfile

Precond.: $\exists S \in SF | S.ServiceID = ServiceID$
Input: ServiceID, goal and profile
Output: —
Effects: $\exists S \in SF | S.ServiceID = ServiceID \wedge S.ServGoal = ServGoal \wedge S.ServProfile = ServProfile$

ModifyProfile: it is used for modifying the description (profile) of a registered service. The client specifies the part of the service to be modified (the goal or the profile). The service ID will not change.

Service 4 ModifyProcess

Precond.: $\exists S \in SF | S.ServiceID = ServiceID \wedge \exists! P \in S.Providers \wedge P.ProviderID = ProviderID$
Input: ServImpID, process and grounding
Output: —
Effects: $\exists P \in Providers | P.ServiceImpID = ServImpID \wedge \exists! P \in S.Providers \wedge P.ProviderID = ProviderID \wedge P.ServProcess = ServProcess \wedge P.ServGroun = ServGroun$

ModifyProcess: it is used for modifying the implementation of a registered service. The client specifies the part of the service to be modified. The service ID will not change. If more than one provider implements the service, then the implementation will not be modified.

Service 5 DeregisterProfile

Precond.: $\exists S \in SF | S.ServiceID = ServiceID$
Input: a valid service ID
Output: —
Effects: $\nexists S \in SF | S.ServiceID = ServiceID$

DeregisterProfile: it is used for deleting a service description.

Service 6 AddProvider
Precond.: $\exists P \in Providers | P.ServiceImpID = ServImpID \wedge ProviderID \notin P.ProvIDList$
Input: IDs of the service (*ServImpID*) and the provider (*ProviderID*)
Output: —
Effects: $\exists P \in Providers | P.ServImpID = ServImpID \wedge ProviderID \in P.ProvIDList$

AddProvider: adds a new provider to an existing service implementation.

Service 7 RemoveProvider
Precond.: $\exists P \in Providers | P.ServImpID = ServImpID \wedge ProviderID \in P.ProvIDList$
Input: IDs of the service (*ServImpID*) and the provider (*ProviderID*)
Output: —
Effects:

1. $\exists P \in Providers | P.ServImpID = ServImpID \wedge ProviderID \notin P.ProvIDList$
2. $\exists P \in Providers | P.ProvIDList = \emptyset \rightarrow [ModifyProcess(P.ServImpID, \emptyset, \emptyset, \emptyset)]$
3. $\exists S \in SF | S.Providers = \emptyset \rightarrow [Deregister(S.ServiceID)]$

RemoveProvider: it deletes a provider from a service implementation. If it is the last provider, then the service implementation is automatically erased. Furthermore, if that is the unique implementation of the service, then the provider is alerted and it can deregister the service.

Service 8 SearchService
Precond.: —
Input: ServicePurpose
Output: list of tuples $< ServiceID, Ranking >$
Effects: —

SearchService: it searches a service whose description satisfies the client request. The search process can use matchmaking, composition and other techniques to solve complex queries. To request a service, the user has to specify a *ServicePurpose*. It is a general structure in which the request is stored. It can be expressed as a *ServiceGoal*, a partial *ServiceProfile* description or a combination of both. The result of the search is a list of tuples $< ServiceID, Ranking >$, where ranking indicates the matching degree between the service and the request.

Service 9 GetProfile
Precond.: $\exists serv \in SF \mid serv.ServiceID = ServiceID$
Input: a valid service ID
Output: service profile and goal
Effects: —

GetProfile: it is used to retrieve the profile details (description) for an specific service.

Service 10 GetProcess
Precond.: $\exists serv \in SF \mid serv.ServiceID = ServiceID$
Input: a valid service ID
Output: a ProvidersList that contains service implementation details
Effects: —

GetProcess: it is used to retrieve the process details (implementation) for an specific service.

The usage of SF meta-services is explained in the following example (see Figure 4.1). In a book selling scenario, there are two providers ($A1$ and $A2$) that can sell books. $A1$ registers a new service profile in the SF called *SellBook Profile* (message 1). The SF identifies this service as $S4$ (message 2). Then agent $A1$ declares itself as a provider for this new service with its particular service process and grounding (message 3). The SF registers this service implementation description with $SI24$ identifier (message 4). Lately, agent $A2$ adds itself as an additional provider for service *SellBook Profile*. It represents an alternative provider with the same process and grounding as $A1$, so it just adds itself as a provider of an existing service implementation (message 5). Finally, a client searches for a low cost book seller (message 6). The SF provides a ranked list with each *ServiceID* and its corresponding matching grade (message 7). With this information the client can retrieve all the service details through *GetProfile* and *GetProcess* meta-services.

4.3 Service Facilitator: Low-Level Design

From the point of view of the implementation, the SF module has two main elements: the SF Agent and the SF services (Figure 4.2) . SF services are the web services that implements the services offered by the platform. SF agent is an intermediate agent and acts as a gateway between agents and web services. This agent receives service requests through FIPA request protocol and it is responsible of making the request to the web service. In the next sections the different technologies and languages used to implement the SF components are described.

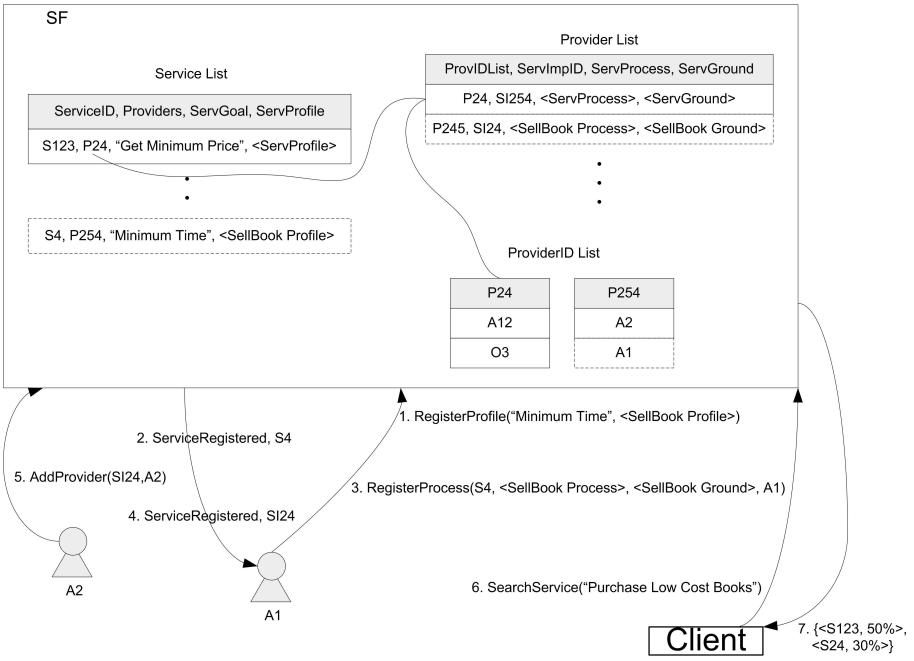

Figure 4.1: Example of SF usage

4.3.1 Services

Service Description

The SF Services are implemented as web services. Each service has a service description in OWL-S and a description in WSDL. The Web Services Description Language (WSDL)[78] is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. OWL-S [57] is a OWL-based web service ontology, which supplies web service providers with a core set of markup language constructs for describing the properties and capabilities of their web services in unambiguous, computer-interpretable form. OWL-S markup of web services will facilitate the automation of web service tasks, including automated web service discovery, execution, composition and interoperability. OWL-S allows for the description of a web service in terms of a Profile, which tells "what the service does", a Process Model, which tells "how the service works", and a Grounding, which tells "how to access the service". The Profile and Process Model are considered to be abstract specifications, in the sense that they do not specify the details of particular message formats, protocols, and network addresses by which a

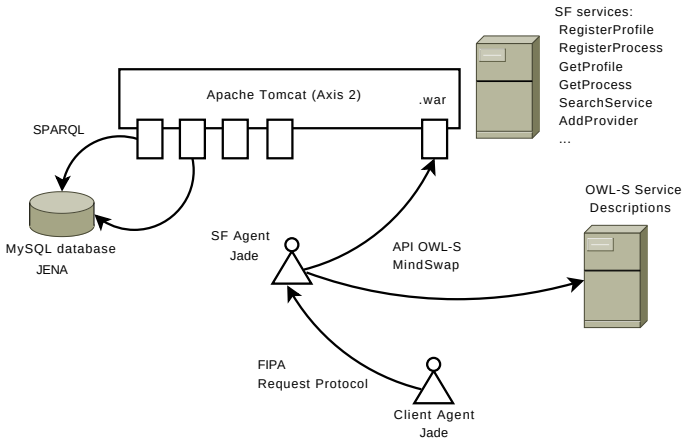

Figure 4.2: Service Facilitator

Web service is instantiated. The role of the grounding is to provide these more concrete details. For each service in the SF should be two OWL-S documents, one with the profile description (Anexo A) and the other with the process and grounding description (Anexo A). The reason to divide the service description in two files is to avoid redundant information in the SF. In some situations could be a service which is provided by two providers. Both providers offer the service with the same IO parameters (same profile) but different implementation (so different process model and grounding)(Figure 4.3).

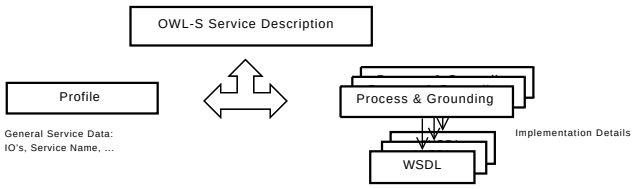

Figure 4.3: Service Description in OWL-S

The WSDL language, developed independently of OWL-S, provides a well developed means of specifying these kinds of details, and has already acquired considerable visibility within the commercial web services community. Therefore, the authors of OWL-S have chosen to define conventions for using WSDL to ground OWL-S services. These conventions are based upon the observation that OWL-S' concept of grounding is generally consistent with WSDL's concept of binding (Anexo A).

Service Development

The runtime chosen for SF web service is Axis2[60]. Based on the Axis2 architecture, there are two implementations of the Apache Axis2 Web services engine - Apache Axis2/Java and Apache Axis2/C. We have decided to use the Apache Axis2/Java due to there are a lot of utilities related with web services which are developed in Java. Apache Axis2 is the core engine for web services. It is a complete re-design and re-write of the widely used Apache Axis SOAP stack, built on the lessons learnt from Apache Axis. Apache Axis2 is more efficient, more modular and more XML-oriented than the older version. It is carefully designed to support the easy addition of plug-in "modules" that extend their functionality for features such as security and reliability.

The services have been developed using the Eclipse Web Tools Platform (WTP) which extends the Eclipse platform with tools for developing web and Java EE applications. The WTP platform contains tools for developing and interacting with Java web services. It consists of:

- web services wizards for creating web service and web services client wizards for consuming Web service
- web services Ant tasks for creating and consuming web services
- wizard extensions for the Apache Axis v1.4 and Apache Axis2 web service runtimes.

To develop the SF services the following step have been followed (Figure 4.4):

1. Generate service descriptions in OWL-S and its correspondent WSDL document.
2. With the Eclipse WTP a web service can be generated automatically from the WSDL file.
3. To add the service logic, the Java implementation file with the skeleton of the service must be modified.
4. .WAR file should be generated to export the service
5. Finally the web service .WAR file have to be located in the Apache Tomcat.

All of the SF web services manipulate semantic information in OWL or service descriptions in OWL-S. To manage and keep all these semantic data in OWL we have used JENA[21]. Jena is a Java framework for building semantic web applications. It provides a programmatic environment for RDF, RDFS and OWL, SPARQL and includes a rule-based inference engine. Jena provides an implementation of the RDF model interface that stores the triples persistently in a database. Each triple is an arc in an RDF model which is called a statement. Each statement asserts a fact about a resource. A statement has three parts :

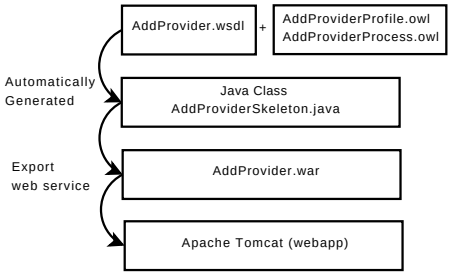

Figure 4.4: Steps for AddProvider Service Implementation

- the subject is the resource from which the arc leaves
- the predicate is the property that labels the arc
- the object is the resource or literal pointed to by the arc

In Figure 4.6 and 4.5 there is an example of some of the statements kept in the SF database. These statements are from a registered *search hotel* service profile.

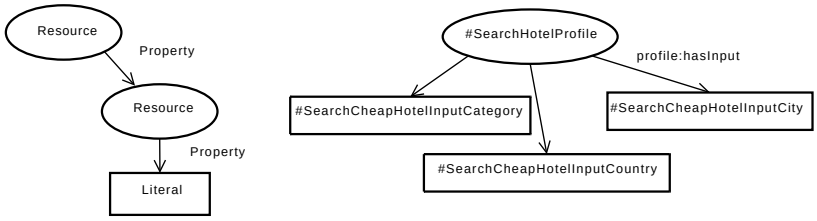

Figure 4.5: RDF graph

This saves the overhead of loading the model each time, and means that you can store RDF models significantly larger than the computer's main memory, but at the expense of a higher overhead (a database interaction) to retrieve and update RDF data from the model. The database used to keep the OWL data is MySQL, an open-source SQL database system available without fee under GPL (Gnu General Public License). It combines good performance with a wide feature set and comes in a variety of configurations to support difference application requirements.

As a query language SPARQL[67] have been used. SPARQL is "data-oriented" in that it only queries the information held in the models; there is no inference in the query language itself. The Jena model may be 'smart' in that it provides the impression that certain triples exist by creating them on-demand, including OWL reasoning. SPARQL does not do anything other than take the description of what the application wants, in the form of a query, and

```
<http://.../SearchCheapHotelProfile.owl#SearchCheapHotelProfile>;
  a       profile:Profile ;
  profile:contactInformation      mind:ProviderA ;
  profile:hasInput
   <http://.../SearchCheapHotelProfile.owl#SearchCheapHotelInputCity>;,
   <http://.../SearchCheapHotelProfile.owl#SearchCheapHotelInputCategory>;,
   <http://.../SearchCheapHotelProfile.owl#SearchCheapHotelInputCountry>;
;
  profile:hasOutput
   <http://.../SearchCheapHotelProfile.owl#SearchCheapHotelOutputHotel>;,
   <http://.../SearchCheapHotelProfile.owl#SearchCheapHotelOutputHotelCompany>;
;
  profile:serviceName "SearchCheapHotel"@en ;
  service:isPresentedBy
   <http://.../SearchCheapHotelProfile.owl#SearchCheapHotelService>;
.
```

Figure 4.6: RDF triples

returns that information, in the form of a set of bindings or an RDF graph. In Figure 4.7 there is an example of a SPARQL query about the service profile of a service with the name *"SearchCheapHotel"@en*.

```
PREFIX  profile: <http://www.daml.org/services/owl-s/1.1/Profile.owl#>;
SELECT  ?x
WHERE
  { ?x  profile:serviceName  "SearchCheapHotel"@en .}

Query Result
ServiceID:
http://.../SF/OWLS/SearchCheapHotelProfile.owl#SearchCheapHotelProfile
Profile http://.../SF/OWLS/SearchCheapHotelProfile.owl
```

Figure 4.7: SPARQL Query

4.3.2 SF Agent Implementation

The SF has been developed as a JADE agent. To implement the logic of the agent we have used the OWL-S API provided by Mindswap. OWL-S API provides a Java API for programmatic access to read, execute and write OWL-S service descriptions. When a FIPA request client message arrives at the SF agent it uses the OWL-S API to access to the service description in OWL and execute the web service.

Executing a service means executing the process it has. The process should have a valid grounding specification in order to invoke the service successfully.

The WSDL and groundings are supported by the API. A process is executed by the *ProcessExecutionEngine.execute(Process, ValueMap)* function where second parameter specifies the values for input parameters. This function returns another ValueMap which contains the output value bindings.

The usage of SF web services is explained in the following example (see Figure 4.10). In a book selling scenario, there are two providers ($A1$ and $A2$) that can sell books. These two agents are JADE agents. $A1$ registers a new service profile in the SF called *SellBook Profile* following the FIPA Request Protocol, so the first message is a *request message*(message 1) (Figure 4.8). The

```
(REQUEST
 :sender  ( agent-identifier :name A1@paracetamol:1099/JADE
 :addresses (sequence http://...:7778/acc ))
 :receiver  (set ( agent-identifier :name SF@paracetamol:1099/JADE ) )
 :content  "http://.../SF/OWLS/RegisterProfileProcess.owl
  RegisterProfileInputServiceGoal=SellBook
  RegisterProfileInputServiceProfile=http://.../SellBookProfile.owl#SellBookProfile"
)
```

Figure 4.8: Message 1

SF intermediate agent is waiting for requests. If SF agent receives a request from a client and can provide the required service it sends an *agree message* (message 2). In this situation, SF agent gets the message content and read the process service description to get the information to execute the service (process and input values). To execute the service the SF uses the OWL-S API. The *RegisterProfile* web service starts its execution. Basically, the service keeps the service profile in the database and return the *service ID.* The SF agent gets the answer from the web service and creates an *inform message* (message 3) to send the returned values.

```
(INFORM
 :sender  ( agent-identifier :name SF@paracetamol:1099/JADE
 :addresses (sequence http://...:7778/acc ))
 :receiver  (set ( agent-identifier :name A1@paracetamol:1099/JADE
 :addresses (sequence http://...:7778/acc )) )
 :content  "RegisterProfileProcess=
   [1,http://...:8080/SF/OWLS/SellBookProfile.owl#SellBookProfile]"
 :reply-with  A1@paracetamol:1099/JADE1221060273666 )
```

Figure 4.9: Message 3

Then agent $A1$ declares itself as a provider for this new service with its particular service process and grounding. To do that the conversation follows the FIPA request protocol. The agent send a *request message* (message 4) to the SF agent intermediator. If the SF agent can execute the service it sends an

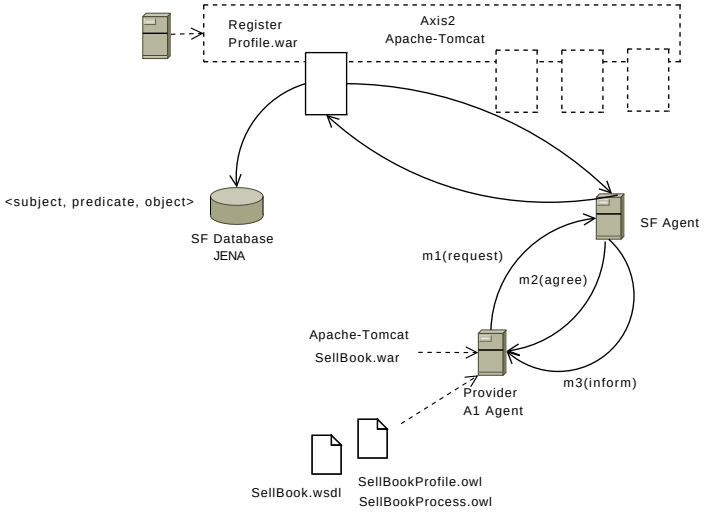

Figure 4.10: Register Profile

inform message (message 5) and executes the *RegisterProcess* web service. The web service *RegisterProcess* loads the process description into the SF database. The data loaded is kept in RDF triples. The answer obtained is sent to the agent $A1$ in an *inform message* (message 4). .

Lately, agent $A2$ adds itself as an additional provider for service *SellBook Profile*. It represents an alternative provider with the same process and grounding as $A1$, so it just adds itself as a provider of an existing service implementation. The protocol that the conversation follows is the FIPA Request, as in the previous conversations.

Finally, a client searches for a low cost book seller. The client sends a *request message* requiring the web service *SearchService*. The SF agent sends to the client an *agree message* in the case that it can make the request and if the SF can, it executes the service and sends the answer to the client in an *inform message*. The SF provides a list with each *ServiceID* and its corresponding matching grade (message 7). With this information the client can retrieve all the service details through *GetProfile* and *GetProcess* web services (Figure 4.11).

4.4 SF Service Discovery and Composition

Many research efforts tackling Web service composition problem via AI planning have been reported. In general, a planning problem can be described as a five tuple $<S,S0,G,A,G>$, where S is the set of all possible states of the world, $S0$ denotes the initial state of the world, G denotes the goal state of the world

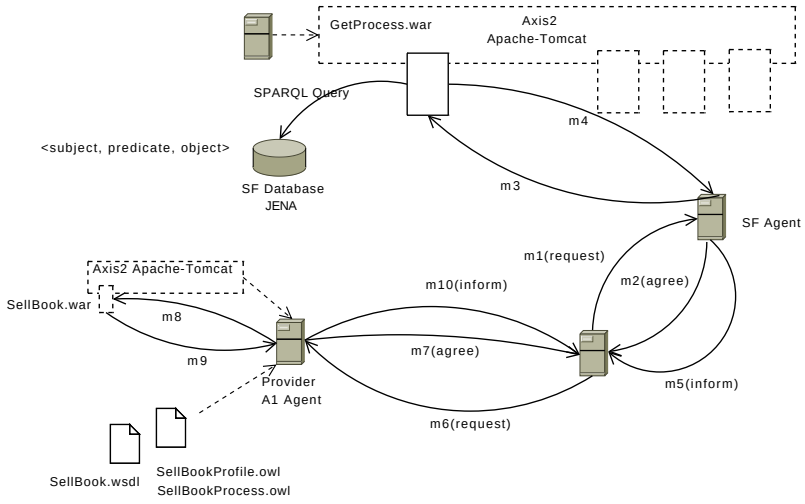

Figure 4.11: Get Process and SellBook

the planning system attempts to reach, A is the set of actions the planner can perform in attempting to change one state to another state in the world, and the translation relation $\Gamma \subseteq G \times A \times S$ defines the precondition and effects for the execution of each action. In the terms of web services, S0 and G are the initial states and the goal states specified in the requirement of Web service requesters. A is a set of available services. G further denotes the state change function of each service.

PDDL is widely recognized as a standardized input for state-of-the-art planners. Moreover since OWL-S has been strongly influenced by PDDL language, mapping from one representation to another is straightforward (as long as only declarative information is considered). When planning for service composition is needed, OWL-S descriptions could be translated to PDDL format [62].

OWL-S service descriptions have not temporal labels in its preconditions and effects. To add this temporal annotations we have to consider that they are represented as logical formulas. Getting logical formulas into RDF is not easy, but is reasonably clear how to proceed. There are actually several possible approaches, depending on how close to RDF/OWL one wants to remain. The key idea in OWL-S is to treat expressions as literals, either string literals or XML literals. The latter case is used for languages such as PDDXML whose standard encoding is in XML. The former case is for other languages such as KIF[30] or PDDL. In our proposal PDDXML, which is based on PDDL[1], has been chosen to express the content of temporal annotated preconditions and effects. In the following sections is explained how PDDL 2.1 actions are annotated and how annotate an OWL-S service taking as reference PDDL 2.1.

4.4.1 Temporal Service Specification

PDDL2.1 has been designed to be backward compatible with the fragment of PDDL that has been in common usage since 1998[24]. This compatibility supports the development of resources which help to establish a scientific foundation for the field of AI planning. PDDL2.1 extends PDDL with numeric and durative extensions to achieve the additional expressive power.

The modelling of temporal relationships in a discretized durative action is done by means of temporally annotated conditions and effects. All conditions and effects of durative actions must be temporally annotated.

Conditions. The annotation of a condition makes explicit whether the associated proposition must hold:

- at the *start* of the interval (the point at which the action is applied)
- at the *end* of the interval (the point at which the final effects of the action are asserted)
- *over* the interval from the start to the end (invariant over the duration of the action)

Invariant conditions in a durative action are required to hold over an interval that is open at both ends (starting and ending at the end points of the action). These are expressed using the *over all*. If one wants to specify that a fact p holds in the closed interval over the duration of a durative action, then three conditions are required: *(at start p)*, *(over all p)* and *(at end p)*.

Effects. The annotation of an effect makes explicit whether the effect is immediate (it happens at the start of the interval) or delayed (it happens at the end of the interval). No other time points are accessible, so all discrete activity takes place at the identified start and end points of the actions in the plan.

Duration. The action duration is represented by a variable called *duration* that represents the durative interval.

OWL-S has been strongly influenced by PDDL language, mapping from one representation to another is straightforward (as long as only declarative information is considered).

In our proposal, all the temporal annotations in preconditions and effects of a OWL-S service description are inside a PDDXML expression. PDDXML is a XML dialect of PDDL that simplifies parsing, reading, and communication PDDL descriptions using SOAP[40]. We have extended this XML language with new labels to facilitate the temporal annotation in the OWL-S service descriptions.

Preconditions and Effects. The OWL-S pre-conditions and effects have been temporally annotated with the same labels that appear in a PDDL domain def-

inition: *at start*, *at end* and *overall* (Figure 4.12).

```
<process:hasPrecondition>
  <pddxml:PDDXML-Condition rdf:ID="PDDXML-Precondition">
   <expr:expressionBody rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    <and>
     <atStart>
      <not>
       <pred name="agentHasKnowledgeAbout">
        <param>?http://.../Packing/GetItems.owl#GetItemsOutputFinishEvent
        </param>
       </pred>
      </not>
     </atStart>
    </and>
   </expr:expressionBody>
  </pddxml:PDDXML-Condition>
 </process:hasPrecondition>
```

Figure 4.12: Precondition with temporal label

Inputs and Outputs. The OWL-S inputs and outputs are not temporally annotated, but if the service is consider as a durative action all input parameters are considered as if they were annotated with the label *at start*. The case of the outputs is similar, but the parameters are considered as if they were annotated with the *at end* label.

Duration. A new non-functional parameter in service description to specify service duration can be specified. In Figure 4.13 there is an example of how specify the service duration.

```
<Duration_param:hasLocal>
  <duration:Duration-Expression rdf:ID="PDDXML-Duration">
   <expr:expressionBody rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    <and>
     <equals>
      <variable><var type="object">?duration</var></variable>
      <constant><const type="int">8</const></constant>
     </equals>
    </and>
   </expr:expressionBody>
  </duration:Duration-Expression>
 </Duration_param:hasLocal>
```

Figure 4.13: Non-Functional Parameter Duration

4.4.2 Temporal Service Composition

The proposal presented allows temporal service composition. Each service is represented as a durative action in PDDL 2.1, so any planner that deals with that language can be used.

The conversion process from OWL-S to PDDL takes a set of available OWL-S temporal service descriptions, a domain description and a planning query as input. The domain description and the planning query contain OWL individuals (facts) which are true initially or are to be achieved by the plan, respectively. They also contain the necessary OWL ontologies. The process result is a plan sequence, i.e a composite service, which satisfies the planning query considering temporal annotations.

For this purpose, the composing process is divided in three stages:

- *From OWL-S to PDDXML* converts the domain ontology and service descriptions in OWL and OWL-S, respectively, to an intermediate language in XML. This stage is an extension of the converter presented in [40]. The converter presented in [40] converts OWL-S documents with PDDXML expressions in service preconditions and effects, but neither consider temporal annotations nor the duration non-functional parameter, so time annotations can not be considered in the service composition process.

- *From PDDXML to PDDL 2.1* is a process in which a parser translate the domain and problem specification in PDDXML in an equivalent PDDL 2.1 problem and domain descriptions (Figure 4.14).

- *Planner* deals with PDDL 2.1 language and it is used to obtain a sequence of durative-actions that represent the service composition.

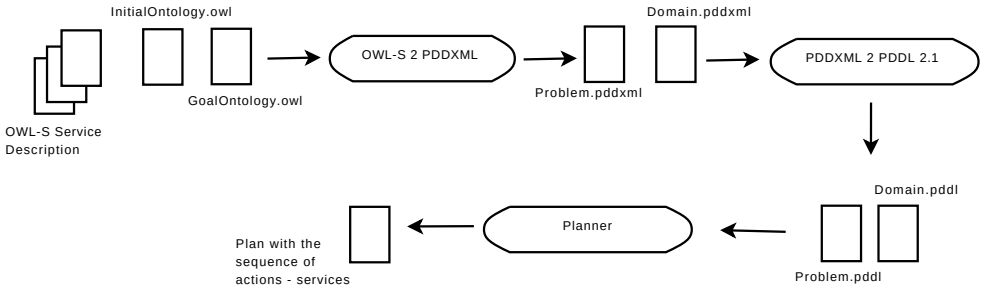

Figure 4.14: Process From OWL-S to PDDL 2.1

4.4.3 PDDL Specification from an OWL-S Description

PDDXML is a XML dialect of PDDL that simplifies parsing, reading, and communication PDDL descriptions using SOAP[40]. The conversion of OWL-S 1.1

service descriptions to PDDXML requires the transcription of types and properties to PDDL predicates as well as the mapping of services to actions.

Any OWL-S service profile *input* parameter correlates with an equally named one of a PDDL action, and the *hasPrecondition* service parameter can directly be transformed to the precondition of the action by use of predicates. The same holds for the *hasEffect* condition parameter. For the conversion of the *output* of an individual OWL-S service to PDDL, the service output parameter is mapped to a special type of the service *hasEffect* parameter. This is because the service *hasEffect* condition explicitly describes how the world state will change while this is not necessarily the case for a *hasOutput* parameter value, though it could implicitly influence the composition planning process. However, PDDL does not allow describing such non-physical knowledge. This problem can be solved by mapping the service output parameter X to a special type of the service *hasEffect* parameter. In particular, every output variable X is described in, and added to the current(physical) planning world state by means of a newly created add-effect predicate in PDDL uniquely named *"agentHasKnowledgeAbout(X)"*. Similarly, each input variable Y is mapped to an input parameter Y of an PDDL action complemented by precondition predicate *"agentHasKnowledgeAbout(Y)"* (Table 4.1). In Appendix A, there is a service description in OWL-S and in Appendix B the result of service conversion in PDDXML.

OWL-S 1.1	PDDL 2.1
hasPrecondition parameter	precondition predicate
hasEffect parameter	effect predicate
hasInput parameter	input predicate + additional precondition agentHasKnowledgeAbout(Input param)
hasOutput parameter	effect predicate + agentHasKnowledgeAbout(Output param)

Table 4.1: Mapping between OWL-S service and PDDL action description

The second stage of the process is the conversion from PDDXML to PDDL 2.1. The transcription of the domain and the problem from PDDXML to PDDL is direct. Once the PDDL files are obtained, the last stage is use a planner to get the plan, the sequence of services, that is needed to achieve the user goals.

4.5 Conclusions

In this chapter the service facilitator has been described with more detail due to is the component that is closely related with the objectives of this work. The SF components have been described from a high level explaining the functionality of each service that it offers. Furthermore a description from a low level is provided. This description shows the technologies used to develop the SF web services and the SF intermediate agent. An example of an initial version is

```
(:durative-action GetOrderService
 :parameters (
?GetOrderInputNotificationEvent - object
?GetOrderOutputItemTypeList - object
?GetOrderOutputOrderCode - object )
 :duration ( = ?duration 4 )
 :condition (and
            (at start (agentHasKnowledgeAbout ?GetOrderInputNotificationEvent ))
            (at start (NotificationEvent ?GetOrderInputNotificationEvent ))
            (at start (ItemTypeList ?GetOrderOutputItemTypeList ))
            (at start (OrderCode ?GetOrderOutputOrderCode ))
            (at start (not(agentHasKnowledgeAbout ?GetOrderOutputOrderCode)))
            )
  :effect (and
          (at end (agentHasKnowledgeAbout ?GetOrderOutputItemTypeList ))
          (at end (identity ?GetOrderOutputItemTypeList ?GetOrderOutputItemTypeList ))
          (at end (agentHasKnowledgeAbout ?GetOrderOutputOrderCode ))
          (at end (identity ?GetOrderOutputOrderCode ?GetOrderOutputOrderCode ))
          )
)
```

Figure 4.15: Durative-Action GetOrderService

provided. This first version has been implemented with the aim of trying the functionality of the SF web services. This version is available and a demo example is also provided.

Besides the service facilitator description a new approach for composing web services based on temporal annotations of services to be included in the SF functionality is provided. The presented procedure exploits numeric and temporal function supporting planers to build the composition. It also provides the facility to generate compositions of web services by using existing PDDL planners such as LPG. To facilitate this task, an extension to OWL-S to include temporal qualified services by including duration as a non-functional parameter and temporal constraints in the preconditions and effects of the service has been developed.

Chapter 5

Example: Packing a Box

5.1 Introduction

There are some environments where the time takes more importance. One of this environments is the manufacturing system in which each process stage has a controlled execution time. In this section a problem in this kind of environment is presented. The problem is in a a packing cell that provides gift boxes. The actions in this cell are presented as services and the problem presented is find the less time consuming service composition to answer a rush order.

5.2 Problem Description

A packing cell is a group of automated machinery and robots. This system enables the customer to select any three of four types of personal grooming items (e.g. razor, shaving gel, deodorant or shaving foam), and pack them into gift boxes. There are two classes of box: a 3-in-a-row, and a T-shape, each with a different configuration of slots where items can be placed. The system is intended to be responsive to changes in the requirements of customers orders, and to be robust to hardware alterations.

A three-loop conveyor system is used to transport shuttles around the cell. Shuttles hold batches of raw materials and the boxes into which items are placed. The track (with its independently controlled gates) provides a high degree of responsive behavior by switching gates to control the flow of shuttles. Shuttles always move forwards along the track unless they touch stop-dogs on the track near the gates and docking stations (which the PLC controls to let the shuttles move when ready) or when their infra-red sensors indicate that another shuttle is in close proximity. There are also two docking stations where shuttles are held, so the robot can process a shuttle's contents: pack item into boxes, unload raw materials into the item storage zone, and unpack items if the box is no longer required for an order (Figure 5.1).

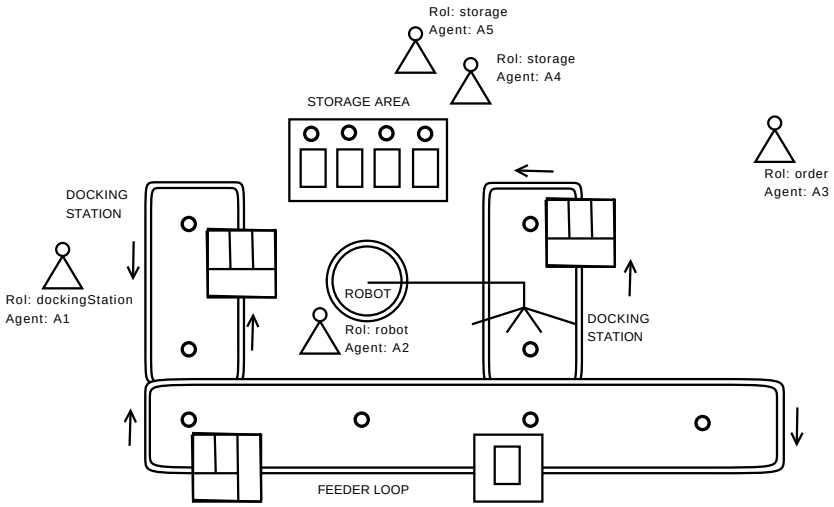

Figure 5.1: Packing Cell

5.3 Packing Box Model with THOMAS

In this example the packing cell is going to be modelled as a organization unit (*PackingCell*). In the organization unit there are four types of roles *PackingCell*: DockingStation, Robot, Order, Storage. Each role has associated services5.1. Next, each role and its services functionality are explained: .

- DockingStation: This role is responsible of the shuttles that are navigating in the docking station carrier and also controls the Piston that blocks or leaves the shuttles. The services of this role are:
  - LockPiston: senses the arrival of a shuttle, it locks it in place using a pneumatic piston ensuring the position of the shuttle. Once the shuttle has arrived and is locked in, it sends a notification event.
  - UnLockPiston: receives the arrival of a finish event and unlocks the pneumatic piston. This allows the shuttle to continue its its navigation in the package cell.
- Robot: This role is responsible of filling the boxes with the correct items. Besides that, control the robot that takes the items from the storage to the box situated on the shuttle. The services of this role are:
  - GetItemsOp: receives a notification event which means that there is a shuttle ready to be filled, the list of items to put inside the box and the confirmation of the storage items availability. With this information the robot starts to pick up the items form the storage and drop them in the boxes. This service first looks up items in the

storage and if there are not products in it the service looks up items in the tracks. When the robot has filled the box with the items the service sends a finish event.

- GetItems: This service is similar to GetItemsOp but when the robot has to pick up the items, first of all it looks up them in the tracks to clean it. So the process takes more time because the robot has to wait the arrival of a item that is navigating around the cell. The idea of this service is first of all clean the three-loop conveyor system of single products.

- Order: This role controls the arrival of new request for the packing cell and besides that is responsible of sending a message, when the box is packed, indicating that the order has been satisfied. The services of this role are:
  - GetOrder: when a notification event arrives means that there is a shuttle with a box ready to be filled with the products specified in a order. The service query its list of orders and select the first order. The output of the service is the order code and the list of items that appears in it to fill the box.
  - SendOrder: When the packing box process is finished, with the order code this service generates a package code that will identify the package in subsequent stages. Finally, the service answers with the availability or not of the items.

- Storage: The storage controls the items that are available in stock. These items could be inside the storage or navigating around the carriers. The services of this role are:
  - QueryCarriersAndStorage: when receives a list of items query the docking stations and the feeder loop to know it the items required are available. If not, the service query the storage.
  - QueryStorage: this service when receives a list of items query the storage to know it all the items required are available. In the case that the items are not available, the service query if the items are navigating around the cell. Finally, the service answers with the availability or not of the items.

All the agents that belong to the unit organization have to take a role and have to implement the services associated with that role.

5.4 Packing Cell Execution

The *PackingCell* unit should be created by an agent. If an agent wants to get into THOMAS and use its services provided by the OMS and SF should request a role of the organization by default. The services specifications offered by the SF and OMS are described with more detail in [33].

Role	Service	Inputs	Outputs	Dur.
DockingSt.	LockPiston	ShuttleEvent OrderEvent	LockPistonFlag NotificationEvent	3
	UnLockPiston	FinishEvent LockPistonFlag	UnLockPistonFlag	2
Robot.	GetItemsOp	NotificationEvent Material Stock List of Item Types	FinishEvent	5
	GetItems	NotificationEvent Material Stock List of Item Types	FinishEvent	8
Order	GetOrder	NotificationEvent	List of Item Types OrderCode	4
	SendOrder	FinishEvent OrderCode	PackageCode	3
Storage	QueryCarriersAndStorage	List of Item Types	MaterialStock	8
	QueryStorage	List of Item Types	MaterialStock	4

Table 5.1: Available Services in the Organization *PackingCell*

```
AcquireRole(UnitID, RoleID)
AcquireRole("Virtual", "Member")
```

After that, the agent could use the services provided by the OMS and SF. The agent manager have to create the unit *PackingCell*, therefore it should request to the OMS the registration of the unit through the service *RegisterUnit*.

```
RegisterUnit(UnitID, Type, Goal, ParentUnitID)
RegisterUnit("PackingCell", "Team", "Pack", "Virtual")
```

There are four types of roles which interact in the unit *PackingCell*: DockingStation, Robot, Order, Storage. These roles should be registered through the service *RegisterRole* provided by the OMS the agent.

```
RegisterRole(RoleID, UnitID)
RegisterRole("DockingStation", "PackingCell")
RegisterRole("Robot", "PackingCell")
RegisterRole("Order", "PackingCell")
RegisterRole("Storage", "PackingCell")
```

The agents that are going to take part in the organization unit should request a registration as a *PackingCell* members through the service *RegisterAgentRole* provided by the OMS. For example, the agent A1 should send request to the OMS.

```
RegisterAgentRole(AgentID, RoleID, UnitID)
RegisterAgentRole("A1", "DockingStation", "PackingCell")
```

```
RegisterAgentRole("A2", "Robot", "PackingCell")
RegisterAgentRole("A3", "Order", "PackingCell")
RegisterAgentRole("A4", "Storage", "PackingCell")
RegisterAgentRole("A5", "Storage", "PackingCell")
```

After that the agents have to register their services through the SF services: *RegisterProfile* and *RegisterProcess*. For example, the services *QueryCarriersAndStorage* and *QueryStorage* has the same profile (same inputs and outputs) but the implementation of the service is different. For this reason, there are only one *RegisterProfile* and two *RegisterProcess*

```
RegisterProfile(ServiceGoal, ServiceProfile)
RegisterProfile("Query", "QueryProfile")

RegisterProcess(ServiceID, ServiceProcess, ProviderID)
RegisterProcess("QueryCarriersAndStorage",
"QueryCarriersAndStorageProcess", "A4")
RegisterProcess("QueryStorage", "QueryStorageServiceProcess", "A5")
```

In the *PackingCell* organization unit, when an order has been placed, and received, and once the shuttle has navigated to the correct docking station, the box can be packed. The start of this process is based on the recognition of a shuttle arrival event. When agent $A1$ (plays the role "DockingStation") senses the arrival of a shuttle, it locks it in place using a pneumatic piston. This provides a reliable way of ensuring the position of the shuttle. Once the shuttle has arrived and is locked in, agent $A1$ notify the controller agent $A2$ (plays the role "robot"). Agent $A2$ takes different actions depending on whether the shuttle has a box and whether the box is full or not. If there is a box, and the box is empty, it is assumed that this box is ready to be packed. The packing operation currently happens on an "all-or-nothing" basis, i.e. it packs all three items into the box, or none of the items will be packed and the shuttle is released for processing later. Since packing is all or nothing, all items must be found within the storage stacks at the start.

The first step is to ask agent $A3$ (plays the role "order") for the list of item types to be packed into the box. This "bill of materials" is sent to agent $A4$ (plays the role "storage") to see if it can be satisfied. Agent $A4$ does not assign items as yet and this is possibly a limitation of the design. In addition, it was found to be simpler and more robust, but perhaps less efficient, to wait until starting the process of packing a box before detecting and responding to a material shortage. Agent $A4$ flags whether or not the "bill of materials" is completely satisfied by the items available in the stack or on shuttle carriers circulating in the cell. If the items are available, but only on shuttle carriers, these shuttles are sent to an appropriate docking station.

As soon as all items are available, the process of building the gift box begins. For each item type, the first step is to find a particular instance of that type in the stack. When the correct stack is found, items are removed from the bottom of that stack until an item of the right type is found. Note that the item type

can be sensed as it is the first part of the EPC (electronic product code) and is registered by RFID readers that sit at the base of the stack. Also note that the item is logically removed from the base as well as physically removed by sending a message to the robot agent.

This process is to send a completion message to agent *A*3, giving the EPC's of the specific items packed into the box. *A*2 agent also send a message to the docking station to release the shuttle.

In these environments time is important to be taken into account. In some situations the configuration of the assembly line can change with the number of orders or with the order types. For example, a factory can use a configuration when the number of orders is not to high, but when a rush order arrives, the time became important and the service configuration in the assembly line changes. In the example, if a rush order arrives, the *PakingCell* agent *A*3 has to make a new composition of services taking into account the services that are available at that moment and the time of these services because there is a time deadline. To do that *A*3 agent sends a request to the SF agent. The request is for the service *SearchService*. The *A*3 agent is looking for a service which should have the inputs: *ShuttleEvent* and *OrderEvent* and the output: *PackageCode*. The *PackingCell* available services are (Table 5.1).

5.5 Packing Cell Service Composition

In the example, when the agent *A*3 requires the SF service *SearchService* the service composition process starts. First of all the *SearchService* tries to find a service with the required characteristics but there is not a single service that fulfill the *A*3 agent request. In that case the service *SearchService* starts the process to find a composition that satisfies the requirements. This process is explained with more detail next. To start the composition process, it is necessary to have three type of files:

- *Temporal service descriptions in OWL-S*: In Appendix B, there is an example OWL-S service description with the non-functional parameter *duration* and with a temporal label in a precondition.
- *Initial state*: OWL file that is composed of facts which are true initially: shuttle event and order event. This facts are ontology individuals (Appendix B).
- *Goal state*: OWL file in which appears the facts (OWL individuals) that are to be achieved by the plan. In the example, these facts are: package code, notification event and intermediate facts that also have to be achieved by the plan: order code, item type list, material stock, etc. (Appendix B).

With the OWL-S service descriptions and the initial and goal states the conversion process *From OWL-S to PDDXML* starts. As a result of this stage two files are obtained: *problem.pddxml* and *domain.pddxml*:

- *problem.pddxml* contains the objects that are present in the problem instance, the initial state description and the goal that the user has defined in the files *InitialOntology.owl* and *GoalOntology.owl*. In difference to action preconditions, the initial state and goal descriptions should be ground, meaning that all predicate arguments should be object or constant names rather than parameters.

- *domain.pddxml* contains the domain predicates and operators (called actions in PDDL) that represent the services. These actions are durative-actions because the non-functional parameter *duration* and the temporal labels are used (Appendix C).

The input of the stage *From PDDXML to PDDL 2.1* are the PDDXML files generated in the previous stage. These files can not be managed by the planners, so it is necessary to translate them. To that end, we have implemented a parser to translate the PDDXML files into PDDL 2.1 files. The output of this stage is the same planning problem but in PDDL (Appendix D) and can be used by any PDDL planner.

Finally, once the problem and domain files in PDDL 2.1 are generated, the planner takes them as input and obtains, if it is possible, a plan or several plans that contains the sequence or sequences of durative actions (temporal service compositions) to achieve the goal state from the initial state. In the example the planner used is LPG (Local search for Planning Graphs)[28]. LPG is a planner based on local search and planning graphs that handles PDDL2.1 domains involving numerical quantities and durations. In this planner the user can specify the running mode. In this case has been chosen the incremental mode. In this mode LPG-td finds a sequence of solutions. When the number "x" of solutions specified for the incremental mode is greater than 1, each solution computed is an improvement with respect to the previous one (in terms of the plan metric indicated in the problem file).

In this example the number of solutions chosen is two. The first one (Figure 5.2) is a plan of six steps (six services involved), and the duration is 23. In the second solution the plan consists on six steps also, but the duration is 19 (Figure 5.3). Both plans achieve the same goal with the same number of services involved, but in the example, if in the packing cell arrives an order with temporal constrains the manager of the cell has to cope with it changing the configuration to another one. In these case, the best configuration of services is the second one.

5.6 Conclusions

In this chapter an example of the owl-s temporal extension is presented. The example is a manufacturing system, more concretely a packing cell. In these kind of environments there are a high probability of time changing restrictions. In the problem presented the problem is the arrival of a rush order that makes necessary a change in the service configuration to deal with the order in time.

```
Time:<ACTION> [action duration; action cost]
0.0003:   (LOCKPISTONSERVICE ARRIVAL_A ORDER_A LOCKPISTONFLAG_A NOTIFICATIONEVENT_A) [3]
3.0005:   (GETORDERSERVICE NOTIFICATIONEVENT_A ITEMTYPELIST_A ORDERCODE_A) [4]
7.0008:   (QUERYCARRIERSANDSTORAGESERVICE ITEMTYPELIST_A MATERIALSTOCK_A) [8]
15.0010:  (GETITEMSSERVICE MATERIALSTOCK_A ITEMTYPELIST_A NOTIFICATIONEVENT_A
FINISHEVENT_A) [8]
23.0012:  (SENDORDERSERVICE ORDERCODE_A FINISHEVENT_A PACKAGECODE_A) [3]
23.0015:  (UNLOCKPISTONSERVICE LOCKPISTONFLAG_A FINISHEVENT_A UNLOCKPISTONFLAG_A) [2]


Solution number: 1
Actions: 6
Execution cost: 6.00
Duration:23.000
Plan quality:23.000
```

Figure 5.2: Sequence of services of the first plan

```
Time:<ACTION> [action duration; action cost]
0.0003:   (LOCKPISTONSERVICE ARRIVAL_A ORDER_A LOCKPISTONFLAG_A NOTIFICATIONEVENT_A) [3]
3.0005:   (GETORDERSERVICE NOTIFICATIONEVENT_A ITEMTYPELIST_A ORDERCODE_A) [4]
7.0008:   (QUERYSTORAGESERVICE ITEMTYPELIST_A MATERIALSTOCK_A) [4]
11.0010:  (GETITEMSOPSERVICE NOTIFICATIONEVENT_A MATERIALSTOCK_A ITEMTYPELIST_A
FINISHEVENT_A) [5]
16.0012:  (SENDORDERSERVICE ORDERCODE_A FINISHEVENT_A PACKAGECODE_A) [3]
16.0015:  (UNLOCKPISTONSERVICE LOCKPISTONFLAG_A FINISHEVENT_A UNLOCKPISTONFLAG_A) [2]


Solution number: 2
Actions: 6
Execution cost: 6.00
Duration:19.000
Plan quality:19.000
```

Figure 5.3: Sequence of services of the second plan

There are seven services described using the OWL-S temporal extension presented in the previous chapter. In the cell there is a initial configuration of the services that controls the machinery and the different stages of the process, but the time that they spend is not admissible to deal with the arrival of the new order. The manager that controls the activity in the packing cell has to consider a new service configuration. In this situation the service composition taking into account the time became a key to give a solution. In the example there is only seven services, but in a real system the number of services that controls the machinery could be intractable to consider all the services available.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The aim of the service-oriented architectures is build compatible software components that will reduce costs of software systems and at the same time increasing the capabilities of the systems. To achieve these objectives service discovery and composition play an important role. Service composition is a form of reuse pieces of software designed in a standardized manner (and with an appropriate level of granularity) to maximize the opportunities. To build an effective composition, the service designer will need a way of finding the most suitable services to act as composition members. Furthermore, once the composition is completed and deployed, potential consumers of the service representing the composition will benefit from an awareness of its existence, purpose, and capabilities. Service discovery is responsible for publish these new composed services. Service discovery allows the designer to find the most suitable services to act as composition members. Furthermore, service discovery avoids the accidental creation of redundant software and as the number of services grows in size the task of selecting an adequate service can quickly grow tedious if all services that are listed under a certain description have to be compared manually for the final selection. In this work a state of the art in this area is presented. With this review the important information required to take into account in service discovery and composition have been analyzed from the point of view of web services and agents systems. The weak and strong points of the presented algorithms have been also analyzed.

Most of the algorithm are centered in the information provided by the IO's. Furthermore, some algorithms are limited to discover simple services. It does not consider the composition discovery nor the use of different ontologies. Another lack is that the process of matching does not consider parameters related with quality of service (QoS); it is only based on the input and output parameters.

There are some algorithms that, apart from IO's information, take into account preconditions and postconditions (IOPE parameters). Where the conventional IOPE-based matchmaking algorithms are unable to determine suitable matches for a request, is decisive take into account compositions. In some of the presented algorithms important points as crossing ontologies or Quality of Service (QoS) to restrict the set of candidate service providers based on user-specified non-functional attributes are taken into account. In addition, web service discovery algorithms do not consider temporal constraints over services.

THOMAS architecture has been presented. This architecture is presented as an architecture that deals with the necessity of virtual organizations and the integration of agents and services, being agents complex entities that can handle the problem of service discovering and composition in dynamic and changing open environments. In this way, two modules are responsible of managing organizations and dealing with service discovery and composition. The module responsible for the last task is the service facilitator (SF) which is along with a redefinition of the FIPA Directory Facilitator that is able to deal with services in a more elaborated way, following Service Oriented Architectures guidelines. The SF acts as a Discovery and Composition Service but also as gateway and yellow pages manager.

To deal with the SF objectives related with service discovery and composition a new approach for composing services based on temporal annotations of services is provided. The presented procedure exploits numeric and temporal function supporting planers to build the composition. It also provides the facility to generate compositions of web services by using existing PDDL planners such as LPG. To facilitate this task, an extension to OWL-S to include temporal qualified services by including duration as a non-functional parameter and temporal constraints in the preconditions and effects of the service has been developed. An example of application in manufacturing environments has been also described.

6.2 Future Work

As a future work there are several ideas related with the Service Facilitator and with the proposal for service composition. The idea related with the SF is to continue with the development of the SF module in THOMAS Architecture. To do that, the following items are being considered:

- Validate the OWL-S documents of the services that require to be registered in the SF.
- Complete the functionality of the SF web services.
- Use a choreography language based on FIPA protocols for service composition.
- Distribute the SF and develop a method for distributed composition.

- Organize services registered in the SF to restrict the number of services to take into consideration in the discovery and composition process.
- Error handling
- Take into account roles in discovery process

Related with the composition proposal the future work is:

- Integration of this approach with a toolkit related with soft-constrains in real time.
- Analyze different planners to find the most suitable for service composition.

6.3 Publications

- E. Del Val and M. Rebollo
  A SURVEY ON WEB SERVICE DISCOVERING AND COMPOSITION
  Web Information Systems and Technologies(Webist) Vol. I pp. 135-142. (2008)

- E. Del Val and M. Rebollo
  Service Discovery and Composition in Multiagent Systems
  Proceedings of Fifth European Workshop On Multi-Agent Systems (EUMAS 2007) pp. 197-212. (2007)

Appendix

A OWL-S SF Service Descriptions

A1 AddProvider OWL-S Service Profile Description

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE uridef [
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
  ...
  <!ENTITY dir "http://.../AddProvider/services/AddProvider?wsdl">
]>
...
<service:Service rdf:ID="AddProviderService">
        <service:presents rdf:resource="#AddProviderProfile"/>
</service:Service>

<profile:Profile rdf:ID="AddProviderProfile">
    <service:isPresentedBy rdf:resource="#AddProviderService"/>
    <profile:serviceName xml:lang="en">AddProvider</profile:serviceName>
    <profile:hasInput rdf:resource="#AddProviderInputServiceImplementationID"/>
    <profile:hasInput rdf:resource="#AddProviderInputProviderID"/>
    <profile:hasOutput rdf:resource="#AddProviderOutputServiceStatus"/>
</profile:Profile>

</rdf:RDF>
```

A2 AddProvider OWL-S Service Process and Grounding Description

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE uridef [
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY dir "http://paracetamol.dsic.upv.es:8080/AddProvider/services/AddProvider?wsdl">
 ...
]>
...
<service:Service rdf:about="http://.../AddProviderProfile.owl#AddProviderService">
 <service:presents rdf:resource="http://.../AddProviderProfile.owl#AddProviderProfile"/>
 <service:describedBy rdf:resource="#AddProviderProcess"/>
 <service:supports rdf:resource="#AddProviderGrounding"/>
</service:Service>

<profile:Profile rdf:about= "http://.../AddProviderProfile.owl#AddProviderProfile">
 <profile:contactInformation>
  <actor:Actor rdf:ID="Provider1">
   <actor:name>Provider1</actor:name>
  </actor:Actor>
 </profile:contactInformation>
```

```
</profile:Profile>

<process:AtomicProcess rdf:ID="AddProviderProcess">
<service:describes rdf:resource="http://.../AddProviderProfile.owl#AddProviderService"/>
 <process:hasOutput rdf:resource="#AddProviderOutputServiceStatus"/>
 <process:hasInput rdf:resource="#AddProviderInputServiceImplementationID"/>
 <process:hasInput rdf:resource="#AddProviderInputProviderID"/>
</process:AtomicProcess>
...
<process:Input rdf:ID="AddProviderInputServiceImplementationID">
     <rdfs:label>InputServiceImplementationID</rdfs:label>
     <process:parameterType rdf:datatype="&xsd;#anyURI">&xsd;#string</process:parameterType>
</process:Input>
...
<grounding:WsdlGrounding rdf:ID="AddProviderGrounding">
 <service:supportedBy rdf:resource="http://.../AddProviderProfile.owl#AddProviderService"/>
 <grounding:hasAtomicProcessGrounding rdf:resource="#AddProviderWsdlAtomicProcessGrounding"/>
</grounding:WsdlGrounding>

<grounding:WsdlAtomicProcessGrounding rdf:ID="AddProviderWsdlAtomicProcessGrounding">
 <grounding:owlsProcess rdf:resource="#AddProviderProcess"/>
 <grounding:wsdlDocument rdf:datatype="&xsd;anyURI">
  http://.../AddProvider/services/AddProvider?wsdl
 </grounding:wsdlDocument>
 <grounding:wsdlOperation>
  <grounding:WsdlOperationRef>
   <grounding:portType rdf:datatype="&xsd;#anyURI">
   http://.../AddProvider/services/AddProviderOperationsPortType
   </grounding:portType>
   <grounding:operation rdf:datatype="&xsd;#anyURI">
   http://.../AddProvider/services/AddProvider
   </grounding:operation>
  </grounding:WsdlOperationRef>
 </grounding:wsdlOperation>

<grounding:wsdlInputMessage rdf:datatype="&xsd;anyURI">
http://.../AddProvider/services/AddProviderMessage
</grounding:wsdlInputMessage>
...
</grounding:WsdlAtomicProcessGrounding>

</rdf:RDF>
```

A3 AddProvider WSDL

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<wsdl:definitions name="AddProvider" ... >
<wsdl:types>...</wsdl:types>
  <wsdl:message name="AddProviderResponse">...</wsdl:message>
```

```
 <wsdl:message name="AddProviderMessage"> ...</wsdl:message>
 <wsdl:portType name="AddProviderPortType">
  <wsdl:operation name="AddProvider">
   <wsdl:input message="tns:AddProviderMessage" />
   <wsdl:output message="tns:AddProviderResponse" />
  </wsdl:operation>
 </wsdl:portType>
 <wsdl:binding name="AddProviderSOAP" type="tns:AddProviderPortType">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />
  <wsdl:operation name="AddProvider">
   <soap:operation soapAction="http://wtp" />
   <wsdl:input><soap:body use="literal"/></wsdl:input>
   <wsdl:output><soap:body use="literal"/></wsdl:output>
  </wsdl:operation>
 </wsdl:binding>
 <wsdl:service name="AddProvider">
  <wsdl:port name="AddProviderSOAP" binding="tns:AddProviderSOAP">
   <soap:address location="http://158.42.185.224:8080/AddProvider/services/AddProvider"/>
  </wsdl:port>
 </wsdl:service>
</wsdl:definitions>
```

B OWL-S Service Descriptions

B1 GetItems OWL-S Service Description

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE uridef [
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY owl "http://www.w3.org/2002/07/owl#">
  <!ENTITY service "http://www.daml.org/services/owl-s/1.1/Service.owl#">
  <!ENTITY profile "http://www.daml.org/services/owl-s/1.1/Profile.owl#">
  <!ENTITY process "http://www.daml.org/services/owl-s/1.1/Process.owl#">
  <!ENTITY grounding "http://www.daml.org/services/owl-s/1.1/Grounding.owl#">
  ...
]>

...

<service:Service rdf:ID="GetItemsService">
 <service:presents rdf:resource="#GetItemsProfile"/>
 <service:describedBy rdf:resource="#GetItemsProcess"/>
 <service:supports rdf:resource="#GetItemsGrounding"/>
</service:Service>

<profile:Profile rdf:ID="GetItemsProfile">
 <service:isPresentedBy rdf:resource="#GetItemsService"/>
 <profile:serviceName xml:lang="en">GetItems</profile:serviceName>
 <profile:hasInput rdf:resource="#GetItemsInputNotificationEvent"/>
 <profile:hasInput rdf:resource="#GetItemsInputMaterialStock"/>
 <profile:hasInput rdf:resource="#GetItemsInputItemTypeList"/>
 <profile:hasOutput rdf:resource="#GetItemsOutputFinishEvent"/>
</profile:Profile>

<process:AtomicProcess rdf:ID="GetItemsProcess">
 <service:describes rdf:resource="#GetItemsService"/>
 <process:hasInput rdf:resource="#GetItemsInputNotificationEvent"/>
 <process:hasInput rdf:resource="#GetItemsInputMaterialStock"/>
 <process:hasInput rdf:resource="#GetItemsInputItemTypeList"/>
 <process:hasOutput rdf:resource="#GetItemsOutputFinishEvent"/>
 <process:hasPrecondition>
  <pddxml:PDDXML-Condition rdf:ID="PDDXML-Precondition">
   <expr:expressionBody rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    <and>
     <atStart>
      <not>
       <pred name="agentHasKnowledgeAbout">
        <param>?http://.../Packing/GetItems.owl#GetItemsOutputFinishEvent
        </param>
       </pred>
      </not>
```

```
     </atStart>
    </and>
   </expr:expressionBody>
  </pddxml:PDDXML-Condition>
 </process:hasPrecondition>
 <Duration_param:hasLocal>
  <duration:Duration-Expression rdf:ID="PDDXML-Duration">
   <expr:expressionBody rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    <and>
     <equals>
      <variable><var type="object">?duration</var></variable>
      <constant><const type="int">8</const></constant>
     </equals>
    </and>
   </expr:expressionBody>
  </duration:Duration-Expression>
 </Duration_param:hasLocal>
</process:AtomicProcess>

<process:Input rdf:ID="GetItemsInputNotificationEvent">
 <rdfs:label>InputNotificationEvent</rdfs:label>
 <process:parameterType rdf:datatype="&xsd;#anyURI">&serv;NotificationEvent
 </process:parameterType>
</process:Input>
<process:Input rdf:ID="GetItemsInputMaterialStock">
 <rdfs:label>InputMaterialStock</rdfs:label>
 <process:parameterType rdf:datatype="&xsd;#anyURI">&serv;MaterialStock
 </process:parameterType>
</process:Input>
<process:Input rdf:ID="GetItemsInputItemTypeList">
 <rdfs:label>InputItemTypeList</rdfs:label>
 <process:parameterType rdf:datatype="&xsd;#anyURI">&serv;ItemTypeList
 </process:parameterType>
</process:Input>
<process:Output rdf:ID="GetItemsOutputFinishEvent">
 <rdfs:label>OutputFinishEvent</rdfs:label>
 <process:parameterType rdf:datatype="&xsd;#anyURI">&serv;FinishEvent
 </process:parameterType>
</process:Output>

<grounding:WsdlGrounding rdf:ID="GetItemsGrounding">
 <service:supportedBy rdf:resource="http://.../Packing/GetItems.owl#GetItemsService"/>
 <grounding:hasAtomicProcessGrounding rdf:resource="#GetItemsWsdlAtomicProcessGrounding"/>
</grounding:WsdlGrounding>

<grounding:WsdlAtomicProcessGrounding rdf:ID="GetItemsWsdlAtomicProcessGrounding">
 <grounding:owlsProcess rdf:resource="#GetItemsProcess"/>
 <grounding:wsdlDocument rdf:datatype="&xsd;anyURI">
  http://.../GetItems/services/GetItems?wsdl
 </grounding:wsdlDocument>
```

```
<grounding:wsdlOperation>
 <grounding:WsdlOperationRef>
  <grounding:portType rdf:datatype="&xsd;#anyURI">
   http://.../GetItems/services/GetItemsOperationsPortType
  </grounding:portType>
  <grounding:operation rdf:datatype="&xsd;#anyURI">
   http://.../GetItems/services/GetItems
  </grounding:operation>
 </grounding:WsdlOperationRef>
</grounding:wsdlOperation>
<grounding:wsdlInputMessage rdf:datatype="&xsd;anyURI">
 http://.../GetItems/services/GetItemsMessage
</grounding:wsdlInputMessage>
<grounding:wsdlOutputMessage rdf:datatype="&xsd;anyURI">
http://.../GetItems/services/GetItemsResponse
</grounding:wsdlOutputMessage>
<grounding:wsdlInput>
 <grounding:WsdlInputMessageMap>
  <grounding:owlsParameter rdf:resource="#GetItemsInputNotificationEvent"/>
  <grounding:wsdlMessagePart rdf:datatype="&xsd;#anyURI">
   http://.../GetItems/services/NotificationEvent
  </grounding:wsdlMessagePart>
 </grounding:WsdlInputMessageMap>
</grounding:wsdlInput>
<grounding:wsdlInput>
 <grounding:WsdlInputMessageMap>
  <grounding:owlsParameter rdf:resource="#GetItemsInputMaterialStock"/>
  <grounding:wsdlMessagePart rdf:datatype="&xsd;#anyURI">
  http://.../GetItems/services/MaterialStock
  </grounding:wsdlMessagePart>
 </grounding:WsdlInputMessageMap>
</grounding:wsdlInput>
<grounding:wsdlInput>
 <grounding:WsdlInputMessageMap>
  <grounding:owlsParameter rdf:resource="#GetItemsInputItemTypeList"/>
  <grounding:wsdlMessagePart rdf:datatype="&xsd;#anyURI">
  http://paracetamol.dsic.upv.es:8080/GetItems/services/ItemTypeList
  </grounding:wsdlMessagePart>
 </grounding:WsdlInputMessageMap>
</grounding:wsdlInput>
<grounding:wsdlOutput>
 <grounding:WsdlOutputMessageMap>
  <grounding:owlsParameter rdf:resource="#GetItemsOutputFinishEvent"/>
  <grounding:wsdlMessagePart rdf:datatype="&xsd;#anyURI">
  http://paracetamol.dsic.upv.es:8080/GetItems/services/FinishEvent
  </grounding:wsdlMessagePart>
 </grounding:WsdlOutputMessageMap>
</grounding:wsdlOutput>
</grounding:WsdlAtomicProcessGrounding>
```

```
</rdf:RDF>
```

B2 InitialOntology.owl

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://.../Packing/ServiceOntology.owl#"
    xml:base="http://.../Packing/ServiceOntology.owl">
  <owl:Ontology rdf:about=""/>

  <owl:Class rdf:ID="ShuttleEvent"/>
  <owl:Class rdf:ID="LockPistonFlag"/>
  <owl:Class rdf:ID="NotificationEvent"/>
  <owl:Class rdf:ID="OrderEvent"/>
  <owl:Class rdf:ID="ItemTypeList"/>
  <owl:Class rdf:ID="UnLockPistonFlag"/>
  <owl:Class rdf:ID="MaterialStock"/>
  <owl:Class rdf:ID="FinishEvent"/>
  <owl:Class rdf:ID="PackageCode"/>
  <owl:Class rdf:ID="OrderCode"/>

  <ShuttleEvent rdf:ID="arrival_A"/>
  <OrderEvent rdf:ID="Order_A"/>
</rdf:RDF>
```

B3 GoalOntology.owl

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://.../Packing/ServiceOntology.owl#"
  xml:base="http://.../Packing/ServiceOntology.owl">
  <owl:Ontology rdf:about=""/>

  <owl:Class rdf:ID="ShuttleEvent"/>
  <owl:Class rdf:ID="LockPistonFlag"/>
  <owl:Class rdf:ID="NotificationEvent"/>
  <owl:Class rdf:ID="OrderEvent"/>
  <owl:Class rdf:ID="ItemTypeList"/>
  <owl:Class rdf:ID="UnLockPistonFlag"/>
  <owl:Class rdf:ID="MaterialStock"/>
  <owl:Class rdf:ID="FinishEvent"/>
  <owl:Class rdf:ID="PackageCode"/>
```

```
  <owl:Class rdf:ID="OrderCode"/>

<PackageCode rdf:ID="PackageCode_A"/>
<NotificationEvent rdf:ID="NotificationEvent_A"/>
<ItemTypeList rdf:ID="ItemTypeList_A"/>
<PackageCode rdf:ID="PackageCode_A"/>
<MaterialStock rdf:ID="MaterialStock_A"/>
<FinishEvent rdf:ID="FinishEvent_A"/>
<OrderCode rdf:ID="OrderCode_A"/>
<LockPistonFlag rdf:ID="LockPistonFlag_A"/>
<UnLockPistonFlag rdf:ID="UnLockPistonFlag_A"/>

</rdf:RDF>
```

C PDDXML

C1 An Action Description in PDDXML

```
<action name="http://.../Packing/GetItems.owl#GetItemsService">
 <parameters>
  <param type="object">
  ?http://.../Packing/GetItems.owl#GetItemsInputMaterialStock
  </param>
  <param type="object">
  ?http://.../Packing/GetItems.owl#GetItemsInputItemTypeList
  </param>
  <param type="object">
  ?http://.../Packing/GetItems.owl#GetItemsInputNotificationEvent
  </param>
  <param type="object">
  ?http://.../Packing/GetItems.owl#GetItemsOutputFinishEvent
  </param>
 </parameters>
 <duration>
  <and>
   <equals>
    <variable>
     <var type="object">?duration</var>
    </variable>
    <constant>
     <const type="int">8</const>
    </constant>
   </equals>
  </and>
 </duration>
 <precondition>
  <and>
   <pred name="agentHasKnowledgeAbout">
    <param>?http://.../Packing/GetItems.owl#GetItemsInputMaterialStock</param>
   </pred>
   <or>
    <pred name="http://.../Packing/ServiceOntology.owl#MaterialStock">
     <param>?http://.../Packing/GetItems.owl#GetItemsInputMaterialStock</param>
    </pred>
   </or>
   <pred name="agentHasKnowledgeAbout">
    <param>?http://.../Packing/GetItems.owl#GetItemsInputItemTypeList</param>
   </pred>
   <or>
    <pred name="http://.../Packing/ServiceOntology.owl#ItemTypeList">
     <param>?http://.../Packing/GetItems.owl#GetItemsInputItemTypeList</param>
    </pred>
   </or>
   <pred name="agentHasKnowledgeAbout">
```

```
    <param>?http://.../Packing/GetItems.owl#GetItemsInputNotificationEvent</param>
   </pred>
   <or>
    <pred name="http://.../Packing/ServiceOntology.owl#NotificationEvent">
     <param>?http://.../Packing/GetItems.owl#GetItemsInputNotificationEvent</param>
    </pred>
   </or>
   <or>
    <pred name="http://.../Packing/ServiceOntology.owl#FinishEvent">
     <param>?http://.../Packing/GetItems.owl#GetItemsOutputFinishEvent</param>
    </pred>
   </or>
   <atStart>
    <not>
     <pred name="agentHasKnowledgeAbout">
      <param>?http://.../Packing/GetItems.owl#GetItemsOutputFinishEvent</param>
     </pred>
    </not>
   </atStart>
  </and>
 </precondition>
 <effect>
  <and>
   <pred name="agentHasKnowledgeAbout">
    <param>?http://.../Packing/GetItems.owl#GetItemsOutputFinishEvent</param>
   </pred>
   <pred name="identity">
    <param>?http://.../Packing/GetItems.owl#GetItemsOutputFinishEvent</param>
    <param>?http://.../Packing/GetItems.owl#GetItemsOutputFinishEvent</param>
   </pred>
  </and>
 </effect>
</action>

  </define_domain>
```

D PDDL 2.1

D1 Problem.pddl

```
(define (problem problem1)
(:domain  InitialOntology)

(:objects

   UnLockPistonFlag_A
   PackageCode_A
   FinishEvent_A
   MaterialStock_A
   LockPistonFlag_A
   ItemTypeList_A
   arrival_A
   Order_A
   OrderCode_A
   NotificationEvent_A
)
(:init
     (agentHasKnowledgeAbout Order_A )
     (identity Order_A Order_A )
     (OrderEvent Order_A )
     (agentHasKnowledgeAbout arrival_A )
     (identity arrival_A arrival_A )
     (ShuttleEvent arrival_A )
     (identity PackageCode_A PackageCode_A )
     (PackageCode PackageCode_A )
     (identity OrderCode_A OrderCode_A )
     (OrderCode OrderCode_A )
     (identity NotificationEvent_A NotificationEvent_A )
     (NotificationEvent NotificationEvent_A )
     (identity MaterialStock_A MaterialStock_A )
     (MaterialStock MaterialStock_A )
     (identity ItemTypeList_A ItemTypeList_A )
     (ItemTypeList ItemTypeList_A )
     (identity UnLockPistonFlag_A UnLockPistonFlag_A )
     (UnLockPistonFlag UnLockPistonFlag_A )
     (identity FinishEvent_A FinishEvent_A )
     (FinishEvent FinishEvent_A )
     (identity LockPistonFlag_A LockPistonFlag_A )
     (LockPistonFlag LockPistonFlag_A )
)
(:goal (and
     (agentHasKnowledgeAbout PackageCode_A )
     (identity PackageCode_A PackageCode_A )
     (PackageCode PackageCode_A )
     (agentHasKnowledgeAbout OrderCode_A )
     (identity OrderCode_A OrderCode_A )
```

```
     (OrderCode OrderCode_A )
     (agentHasKnowledgeAbout NotificationEvent_A )
     (identity NotificationEvent_A NotificationEvent_A )
     (NotificationEvent NotificationEvent_A )
     (agentHasKnowledgeAbout MaterialStock_A )
     (identity MaterialStock_A MaterialStock_A )
     (MaterialStock MaterialStock_A )
     (agentHasKnowledgeAbout ItemTypeList_A )
     (identity ItemTypeList_A ItemTypeList_A )
     (ItemTypeList ItemTypeList_A )
     (agentHasKnowledgeAbout UnLockPistonFlag_A )
     (identity UnLockPistonFlag_A UnLockPistonFlag_A )
     (UnLockPistonFlag UnLockPistonFlag_A )
     (agentHasKnowledgeAbout FinishEvent_A )
     (identity FinishEvent_A FinishEvent_A )
     (FinishEvent FinishEvent_A )
     (agentHasKnowledgeAbout LockPistonFlag_A )
     (identity LockPistonFlag_A LockPistonFlag_A )
     (LockPistonFlag LockPistonFlag_A )
     )
)
)
```

D2 Domain.pddl

```
(define (domain  InitialOntology)
(:requirements :durative-actions)
(:predicates
     (NotificationEvent ?individual_0 - object )
     (FinishEvent ?individual_0 - object )
     (ShuttleEvent ?individual_0 - object )
     (identity ?individual_0 - object ?individual_1 - object )
     (PackageCode ?individual_0 - object )
     (OrderEvent ?individual_0 - object )
     (agentHasKnowledgeAbout ?object_parameter - object )
     (MaterialStock ?individual_0 - object )
     (UnLockPistonFlag ?individual_0 - object )
     (OrderCode ?individual_0 - object )
     (LockPistonFlag ?individual_0 - object )
     (ItemTypeList ?individual_0 - object )
)

(:durative-action GetOrderService
 :parameters (
?GetOrderInputNotificationEvent - object
?GetOrderOutputItemTypeList - object
?GetOrderOutputOrderCode - object )
 :duration ( = ?duration 4 )
 :condition (and
```

```
            (at start (agentHasKnowledgeAbout ?GetOrderInputNotificationEvent ))
            (at start (NotificationEvent ?GetOrderInputNotificationEvent ))
            (at start (ItemTypeList ?GetOrderOutputItemTypeList ))
            (at start (OrderCode ?GetOrderOutputOrderCode ))
            (at start (not(agentHasKnowledgeAbout ?GetOrderOutputOrderCode)))
            )
  :effect (and
          (at end (agentHasKnowledgeAbout ?GetOrderOutputItemTypeList ))
          (at end (identity ?GetOrderOutputItemTypeList ?GetOrderOutputItemTypeList ))
          (at end (agentHasKnowledgeAbout ?GetOrderOutputOrderCode ))
          (at end (identity ?GetOrderOutputOrderCode ?GetOrderOutputOrderCode ))
          )
)
...
```

Bibliography

[1] Pddl— the planning domain definition language by too many people to mention at too many places to mention.

[2] Web services agent integration project.
URL `http://wsai.sourceforge.net/index.html`

[3] C. Abela, M. Montebello, Daml enabled web services and agents in the semantic web, in: WS–RSD'02, Germany, 2002.

[4] L. Aversano, G. Canfora, A. Ciampi, An algorithm for web service discovery through their composition, in: IEEE International Conference on Web Services (ICWS04), 2004.

[5] D. Bachlechner, K. Siorpaes, H. Lausen, D. Fensel, Web Service Discovery - A Reality Check, 2006.

[6] J. Bailey, Fast discovery of interesting collections of web services, in: WI '06: Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence, IEEE Computer Society, Washington, DC, USA, 2006.

[7] Baldoni, Baroglio, Martelli, Patti, Schifanella, Service selection by choreography-driven matching, in: ECOWS Workshop on Emerging Web Services Technology, 2007.

[8] M. Baldoni, C. Baroglio, A. Martelli, V. Patti, Reasoning about interaction protocols for customizing web service selection and composition, No. Vol.70, 2007.

[9] Y. Balzer, Improve your soa project plans (2004).

[10] S. Bansal, J. Vidal, Matchmaking of web services based on the daml-s service model, in: Second International Joint Conference on Autonomous Agents (AAMAS03), T. Sandholm and M. Yokoo, 2003.

[11] B. Benatallah, M.-S. Hacid, C. Rey, F. Toumani, Request rewriting-based web service discovery, in: International Semantic Web Conference, 2003.

[12] E. Bircher, T. Braun, An Agent-Based Architecture for Service Discovery and Negotiation in Wireless Networks, 2004.

[13] A. Brogi, S. Corfini, J. Aldana, I. Navas, Automated discovery of compositions of services described with separate ontologies.

[14] A. Brogi, S. Corfini, J. F. A. Montes, I. N. Delgado, A prototype for discovering compositions of semantic web services., in: G. Tummarello, P. Bouquet, O. Signore (eds.), SWAP, vol. 201 of CEUR Workshop Proceedings, CEUR-WS.org, 2006.

[15] A. Brogi, S. Corfini, R. Popescu, Composition-oriented service discovery, Hakodate, Japan, 2003.

[16] C. Cáceres, A. Fernández, S. Ossowski, M. Vasirani, Role-based service description and discovery, in: International Joint Conference on Autonomous Agents and Multi-Agent Systems, 2006.

[17] J. Cardoso, A. Sheth, Semantic e-workflow composition, Journal of Intelligent Information Systems (JIIS).

[18] I. Constantinescu, B. Faltings, Efficient matchmaking and directory services, Technical Report No IC/2002/77 (2002).

[19] B. Corradini, C. Ercoli, E. Merelli, B. Re, An agent-based matchmaker, in: In proceedings of WOA 2004 dagli Oggetti agli Agenti - Sistemi Complessi e Agenti Razionali, 1988.

[20] J. Dang, M. Hungs, Concurrent Multiple-Issue Negotiation for Internet-Based Services, No. Vol.10 - 6, 2006.

[21] I. Dickinson, Jena ontology api, `http://jena.sourceforge.net/ontology/index.html`.

[22] T. Erl, SOA: Principles of Service Design, 2007.

[23] A. Fernandez, M. Vasirani, C. Caceres, S. Ossowski, An abstract architecture for semantic service coordination in agent-based intelligent peer-to-peer environments, in: poster paper at the ACM SAC Special Track on Coordination Models, Languages and Applications (SAC'06), 2006.

[24] M. Fox, D. Long, Pddl2.1: An extension to pddl for expressing temporal planning domains, J. Artif. Intell. Res. (JAIR) 20 (2003) 61–124.

[25] S. Fronk, I. Jelnek, Semantic mining of web documents, in: In International Conference on Computer Systems and Technologies.

[26] C. Gao, R. Liu, Y. Song, H. Chen, A model checking tool embedded into services composition environment, in: GCC '06: Proceedings of the Fifth International Conference on Grid and Cooperative Computing (GCC'06), IEEE Computer Society, Washington, DC, USA, 2006.

[27] J. Garofalakis, Y. Panagis, E. Sakkopoulos, A. Tsakalidis, Web service discovery mechanisms: Looking for a needle in a haystack?
URL `citeseer.ist.psu.edu/garofalakis04web.html`

[28] A. Gerevini, I. Serina, Lpg: A planner based on local search for planning graphs with action costs., in: AIPS, 2002.

[29] Y. Gil, S. Ramacachandran, Phosphorus: A task-based agent matchmaker, in: Proceedings of the International Conference on Autonomous Agents (Agents'01) (Short paper), 2001.

[30] M. L. Ginsberg, Knowledge interchange format: the kif of death, AI Mag. 12 (3) (1991) 57–63.

[31] F. Giunchiglia, P. Traverso, Planning as model checking, in: ECP, 1999.
URL `citeseer.ist.psu.edu/giunchiglia99planning.html`

[32] D. Greenwood, Jade web service integration gatway (wsig), in: JADE AAMAS Workshop, 2005.

[33] GTI-IA, An abstract architecture for virtual organizations: The thomas project (2007).
URL `http://www.fipa.org/docs/THOMASarchitecture.pdf`

[34] S. Hashemian, F. Mavaddat, A graph-based approach to web services composition, In IEEE Computer Society.

[35] S. Jha, P. Chalasani, O. Shehory, K. Sycara, A formal treatment of distributed matchmaking, in: Proc. of the 2nd Int. Conference on Autonomous Agents, No. Vol.3, 1998.

[36] J.Radatz, M. Sloman, A standard dictionary for computer terminology: Proyect 610, IEEE Computer.

[37] S. Kalepu, S. Krishnaswamy, S. Loke, Reputation = f(user ranking, compliance, verity), in: Proceedings of the IEEE International Conference on Web Services, 2004.

[38] S. KluS, F.Hashemian, A graph-based approach to web services composition, in: Symposium on Applications and the Internet (SAINT'05), 2005.

[39] M. Klusch, B. Fries, K. Sycara, Automated semantic web service discovery with owls-mx, in: Proceedings of 5th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), Hakodate, Japan, 2006.

[40] M. Klusch, A. Gerber, Semantic web service composition planning with owls-xplan, in: In Proceedings of the 1st Int. AAAI Fall Symposium on Agents and the Semantic Web, 2005.

[41] N. Kokash, Web service discovery with implicit qos filtering, in: Proceedings of the IBM PhD Student Symposium, 2005.

[42] U. Kuster, H. Lausen, B. Konig-Ries, Evaluation of semantic service discovery - a survey and directions for future research, in: Proceedings of the 2nd Workshop on Emerging Web Services Technology (WEWST07) in conjunction with the 5th IEEE European Conference on Web Services, Halle (Saale), Germany, 2007.

[43] L. Lei, I. Horrocks, A software framework for matchmaking based on semantic web technology, in: Twelfth International World Wide Conference (WWW2003), Germany, 2003.

[44] S. Ludwig, Weight assignment of semantic match using user values and a fuzzy approach, in: International Conference on Service-Oriented Computing, 2007.

[45] U. Manikrao, T.V.Prabhakar, Dynamic selection of web services with recommendation system., in: Proceedings of the International Conference on Next Generation Web Services Practices (NWESP), No. 117, 2005.

[46] E. Maximilien, M. Singh, A framework and ontology for dynamic web services selection, No. Vol.8 - 5, 2004.

[47] I. Mecar, Agent-oriented semantic discovery and matchmaking of web services, in: 8th International Conference on Telecommunications, 2005.

[48] S. Mokhtar, A. Kaul, N. Georgantas, V. Issarny, Towards efficient matching of semantic web service capabilities, in: Proc. of WS-MaTe06, 2006.

[49] M. Moore, T. Suda, A decentralized and self-organizing discovery mechanism, in: Proc. Of the First Annual Symposium on Autonomous Intelligent Networks and Systems, 2002.

[50] S. Mullender, P. Vitanyi., Distributed Match-Making, No. Vol.3, 1988.

[51] S. Nakajima, Model-checking verification for reliable web service, in: OOPSLA 2002 Workshop on Object-Oriented Web Services, Seattle, Washington, 2002.

[52] T. Nguyen, R. Kowalczyk, Ws2jade: Integrating web service with jade agents, in: Technical Report: SUTICT-TR2005.03, 20 July 2005.

[53] E. Ogston, S. Vassiliadis, Local distributed agent matchmaking, in: Proceedings of the 9th International Conference on Cooperative Information Systems, 2001.

[54] E. Ogston, S. Vassiliadis, Matchmaking among minimal agents without a facilitator, in: Proceedings of the 5th International Conference on Autonomous Agents, 2001.

[55] A. Ouksel, Y. Babad, T. Tesch, Matchmaking software agents in b2b markets, in: Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04), 2004.

[56] S. Overhage, On specifying web services using uddi improvements (2002).

[57] Owl-s: Semantic markup for web services, `http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122`.

[58] M. Paolucci, Semantic matching of web services capabilities, in: The First International Semantic Web Conference, 2002.

[59] J. Pathak, N. Koul, D. Caragea, V. Honavar, A framework for semantic web service discovery, in: WIDM'05, Germany, 2005.

[60] S. Perera, A. Ranabahu, Axis2 - the future of web services, `http://www.jaxmag.com/itr/online_artikel/psecom,id,747,nodeid,147.html`.

[61] S. Prabhu, Towards distributed dynamic web service composition, in: ISADS '07: Proceedings of the Eighth International Symposium on Autonomous Decentralized Systems, IEEE Computer Society, Washington, DC, USA, 2007.

[62] J. Rao, X. Su, A survey of automated web service composition methods, in: LNCS, vol. 3387/2005, Springer, 2005, pp. 43–54.
URL `http://www.springerlink.com/content/4m6w37g0jffk9bv4`

[63] V. J. S. Ossowski, J. Bajo, H. Billhardt, V. Botti, J. Corchado, Open mas for real world applications: an abstract architecture proposal.

[64] M. Sensoy, C. Pembe, H. Zirtiloglu, P. Yolum, A. Bener, Experience-based service provider selection in agent-mediated e-comerce, in: Engineering Applications of Artificial Intelligence, No. 3, 2007.

[65] A. Shaikh, S. Ludwig, O. Rana, A cognitive trust-based approach for web service discovery and selection, in: European Conference on Web Services, 2005.

[66] K. Sigdel, K. Bertels, B. Pourebrahimi, S. Vassiliadis, L. Shuai, A framework for adaptive matchmaking in distributed computing, in: In proceeding of GRID Workshop, 2005.

[67] Sparql query language for rdf, `http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115`.

[68] N. Srinivasan, M. Paolucci, K. Sycara, Adding owl-s to uddi implementation and throughput, in: In Workshop on Semantic Web Service and Web Process Composition, 2004.

[69] L. Steller, S. Krishnaswamy, J. Newmarch, Discovering relevant services in pervasive environments using semantics and context, in: IWUC, 2006.

[70] K. Sycara, M. Klusch, Brokering and matchmaking for coordination of agent societies: A survey, Coordination of Internet Agents: Models, Technologies and Applications (2001) 197–224.

[71] K. Sycara, S. Widoffand, M. Klusch, J. Lu, Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace, Journal on Autonomous Agents and Multi-Agent Systems.

[72] K. Trec, A. Devli, G. Jei, M. Kuek, S. Dei, Semantic matchmaking of advanced personalized mobile services using intelligent agents, in: Proceedings of the 12th International Conference on Software, Telecommunications and Computer Networks (SoftCOM), 2004.

[73] E. D. Val, M. Rebollo, Service discovery and composition in multiagent systems, in: Proceedings of Fifth European Workshop On Multi-Agent Systems (EUMAS 2007), Association Tunisienne D'Intelligence Artificielle, 2007.

[74] E. D. Val, M. Rebollo, A survey on web service discovering and composition, in: Web Information Systems and Technologies(Webist), vol. I, 2008.

[75] M. B. van Riemsdijk, M. Wirsing, Goal-oriented and procedural service orchestration - a formal comparison, in: MALLOW-AWESOME'007, Durham, 2007.

[76] C. Walton, Model checking multi-agent web services, in: Proceedings of the 2004 Spring Symposium on Semantic Web Services, Stanford, CA, USA, 2004.
URL `citeseer.ist.psu.edu/walton04model.html`

[77] B. Wolf-Tilo, W. Matthias, Towards personalized selection of web services, in: The Twelfth International WWW Conference en Budapest, 2003.

[78] Web services description language (wsdl) 1.1, `http://www.w3.org/TR/2001/NOTE-wsdl-20010315`.

[79] L. Yang, B. K. Sarker, V. C. Bhavsar, H. Boley, A weighted-tree simplicity algorithm for similarity matching of partial product descriptions, in: In Proceedings of ISCA 14th International Conference onIntelligent and Adaptive Systemsand Software Engineering, Toronto, 2005.

[80] H. Q. Yu, S. Reiff-Marganiec, Semantic web services composition via planning as model checking, Tech. Report CS-06-003, University of Leicester (2006).

[81] Z. Zhang, C. Zhang, An improvement to matchmaking algorithms for middle agent, in: AAMAS, 2002.