Chapter 8

WEB SERVICES COMPOSITION

Daniela Barreiro Claro[1,2] and Patrick Albers[1] and Jin-Kao Hao[2]
[1]ESEO, 4 rue Merlet de la Boulaye, BP 30926 49009 Angers cedex 01 France. daniela.claro@eseo.fr, patrick.albers@eseo.fr

[2]LERIA, University of Angers, 2 Boulevard Lavoisier, 49045, Angers cedex 01 France. jin-kao.hao@univ-angers.fr

1. INTRODUCTION

Nowadays many enterprises publish their applications functionalities on the Internet. This new generation of applications allows greater efficiency and availability for business. In fact, more and more applications make functionalities available using a web service format.

However there are many services around the web, each one, taken alone, has a limited functionality. In many cases, a single service is not sufficient to respond to the user's request and often services should be combined through services composition to achieve a specific goal. For example, if a user wants to travel, it is not sufficient to book a flight, but she should also take care of reserving a hotel, renting a car, getting entertained, and so on. Such composition is carried out manually today, it means that the user needs to execute all these services one by one and these tasks can be time and effort consuming.

For that reason, the notion of composite services is starting to be used as a collection of services combined to achieve a user's request. In other words, from a user perspective, this composition will continue to be considered as a simple service, even though it is composed of several web services.

Nevertheless, prior to composing web services, candidate services should first be discovered and then selected. One difficulty is that many functionally similar services are available and thus, the number of discovered services by search mechanisms increases as a consequence. The discovery process returns a set of candidate services from which the subset of those belonging to the composition should be extracted according to non-functional criteria (i.e. cost, availability, reputation). In fact, discovery is a prerequisite for selection, but selection is the main problem (Sreenath and Singh 2004). The non-functional criteria are here characterized by the QoS model presented in each web service. The QoS model has more than one criterion to be evaluated. Thus, services composition can be considered as a multiobjective optimization problem.

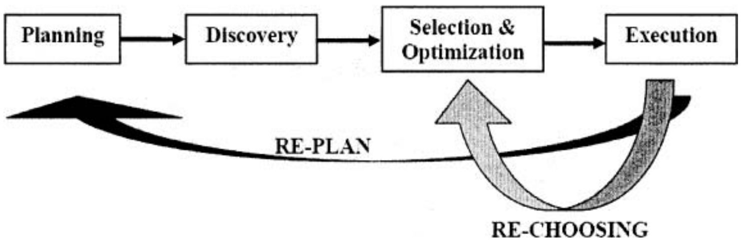


Figure 8-1. SPOC Architecture

As depicted in Figure 8-1, we propose SPOC (**S**emantic based **P**lanning for **O**ptimal web services **C**omposition), an architecture to compose web services. In our point of view, the problem of composing web services can be reduced into four fundamental phases: the first one is planning, which determines the execution order of the tasks, we consider here a task as being a service functionality or a service activity. The second one is discovery that aims at finding candidate services for each task in the plan. The third phase aims at optimizing services composition and is the point treated in this chapter, and, finally, the fourth concerns execution. This fourth phase is characterized as a problem because, even during the execution process, the services may not be found and another tradeoff composition needs to be used or other plan needs to be envisioned.

The composition of web services starts by creating the initial plan based on tasks definition. All the definitions of existing tasks should be located in a repository that the planner can consult for obtaining tasks interfaces. This repository can be represented as an ontology and for us, it can be an improvement over UDDI registries. Hence, we propose a UDDI (Universal Description, Discovery and Integration) that is actually an ontology which describes the services and their providers in an unambiguous way. The name we give to this new UDDI is UDDI-O, standing for ontology. Thus, prior to knowing task interfaces, it is necessary to find a plan that satisfies the users' request. After creating the initial plan, the discovery process will take place. The discovery process aims at matching service descriptions with task definitions that belong to the plan. The present work will not cover the matchmaking problem concerning web services discovery. The optimization phase is the main topic of this chapter and will be explained in detail in the next sections. As a result of this phase we obtain a set of Pareto optimal solutions to execute services composition. In the execution phase, if some service is not available such as an invalid URL or changed location, the environment proposes another Pareto optimal solution to be executed (this corresponds to "re-choosing" in figure 8-1. If after some predefined time the problem continues, the environment will propose to construct another plan, for example, by reordering the tasks (this corresponds to re-plan in Figure 8-1.

This work proposes an analysis of quality criteria in order to select from a set of services those that will belong to the composition. It is organized as follows: the next section describes the selection process and the QoS model. Here, we reinforced the concepts of reputation, because the original concept (Zeng et al. 2003) did not measure the pertinence of the rank given to a service by a user. Thus, in our model, rankings from users with good knowledge of the service domain are considered more accurate. For this purpose, we use fuzzy numbers to measure this criterion. The third section describes web services composition emphasizing its structure and the models that exist to compose web services. The fourth section explains the problem model with its objectives and constraints. In the fifth section we explain the multiobjective approach emphasizing the Pareto and Non-Pareto approach. The sixth section presents existing works related to ours and the seventh highlights our experimentations. We conclude in the last section.

2. WEB SERVICES SELECTION

The current web service architecture and semantic web efforts address the problem of web service discovery but not of web services selection. Discovery deals with finding a set of services that corresponds to a predetermined user request while selection deals with choosing a service between those that are discovered. Moreover, selection seems to be the main problem. In fact, if the discovery process is exhaustive, a very large number of services may be found. Due to the number of services, and consequently the number of candidate services, the selection process will be harder (Sreenath and Singh 2004).

Discovering services mean matching a user request with service functionalities. Works have been undergone concerning service architecture (Sreenath and Singh 2004) in order to better describe web services. Even though more functionalities are incorporated into service descriptions, it still remains difficult for selection to find the subset of services that will be part of the composition (Sreenath and Singh 2004).

Despite the fact that functional attributes have been incorporated by web services architecture, selection should consider more than functional criteria to make a distinction between discovered services. As a result, a quality of service (QoS) model composed of time, cost, availability and reputation is proposed as non-functional criteria. Since non-functional criteria have been incorporated by each service, selection can use these QoS variables in order to choose the optimal subset from all the discovered services.

2.1 QoS (Non-functional) Model

The aim of the selection process is to choose among services discovered according to their functionalities, those that will belong to the composition. The set of discovered services can be subdivided into the subsets of services that are all candidates for a given task. Therefore, in the discovered set, there are subsets of services that execute a determined task and other subsets that execute another kind of tasks. As mentioned earlier, we consider here a task as being a service functionality or a service activity. Thus, in the selection process we should determine a set of candidate services s_i, $i \in [1..n]$ that can execute a set of tasks t_j, $j \in [1..m]$. Our main goal, considering that there is a set of candidate services for each task, is to determine which service fulfills each task, thus finding services composition.

The QoS model that we propose is composed of four criteria as parameters for the quality model: cost, time, availability and reputation. Each of the candidate services will receive a value for representing these quality criteria. Each of these criteria is presented below.

Cost. (Zeng et al. 2003) (Cardoso et al. 2004)(Liu et al. 2004) The cost quality c_{ij} is the amount that a service requester needs to pay to execute service *i* using task *j*:

$$c_{ij}, i \in [1..n], j \in [1..m]$$

We consider that c_{ij} is undetermined when service *i* cannot execute task *t*.

Time. (Zeng et al. 2003) (Cardoso et al. 2004) (Liu et al. 2004) The time quality t_{ij} measures the execution time between the moment the request is sent and the moment the results are received:

$$t_{ij}, i \in [1..n], j \in [1..m]$$

Availability. (Zeng et al. 2003) The availability quality a_{ij} is the probability that the service can be accessed and used. It is a function of the number of times the service responds to a request and of the number of total requests made to the service. We can express by:

$$a_{ij} = \frac{req_{ij}}{tot_{ij}}, tot_{ij} \neq 0, i \in [1..n], j \in [1..m]$$

where req_{ij} is the number of successful requests to service i using task j, and tot_{ij} is the total number of invocations.

Reputation. The reputation quality r_{ij} is the measure of its trustworthiness. It depends on the user's experience using the service. Different end users can have different opinions about the same service.

For many authors (Zeng et al. 2003) (Liu et al. 2004), reputation can be defined as the average ranking given to the service by end users. The reputation of a given service is usually defined as:

$$q_{rep} = \frac{\sum_{b=1}^{N} k_b}{N}$$

where k_b is the b^{th} ranking given to the service and N is the number of times the service has been ranked.

However, there is no consensus concerning measuring reputation. Here, we propose a new way of measuring reputation. We tried to translate a real world judgment into our example. Thus, in real world, when something is judged for example, a paper in a conference, the reviewers have to give their knowledge domain, prior to giving their judgment. In the case where a reviewer receives a paper that she classifies as belonging only 60% to her area (knowledge domain), the grade that is given must be moderated based on 60% of knowledge. If the same grade is given by a reviewer with 90% of know-how on the domain, for sure her grade will be more accurate. Translating this real scenario into our reputation quality, we must have another way to measure reputation, including the knowledge domain of end users. After service execution, the user ranks the service, and gives a percentage about her knowledge on the service's domain. It will be, for instance, a simple question as "how much do I know about this area".

In order to measure this criterion, we used fuzzy logic to represent an imprecise quantity, as "nearly 8" or "practically 15" (Moura 2001). We used the notion of fuzzy number which is represented as

$$\tilde{a} = [\underline{a}, a, \overline{a}]$$

where $\tilde{a}$ is the fuzzy number with minimal limit, modal value and maximal limit respectively. The *linguistic variables* that represent our reputation values are: bad, average and good, as shown in Figure 8-2.

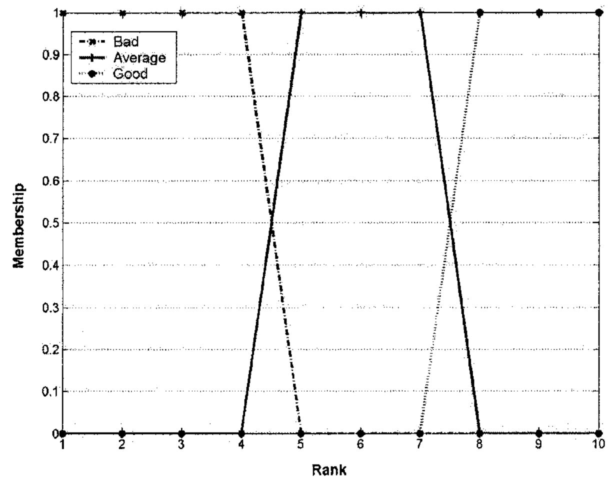


Figure 8-2. Fuzzy set representation

Figure 8-2 shows that until 4, all grades are considered *bad*, from 5 to 7, grades are *average*, and after 8, all grades are *good*. The measure between 4 and 5, for example, depends on membership values. The membership or degree of pertinence means how much a value is inside a set, for example the *bad* set or inside the *average* set. Thus, if a service has a rank of 4.8 we need to analyze its membership $\mu(d1)$. If its membership has the value 0.33, it means that it belongs to the *bad* set. On the other hand, if it has 0.66 as membership value, it belongs to the *average* set. Each service will be ranked several times and thus we will have a set of fuzzy numbers. However, at the end, what we need is a crisp number that characterizes the reputation value, and for that we need to convert fuzzy sets to a crisp number. Defuzzification is the final phase that does this conversion. There are several defuzzification methods, but we use the CENTROID method that calculates the hypothetical center of gravity for the output fuzzy set (Löstedt et al. 2000) (Fuzzy 2005). Thus, our reputation criterion is characterized as:

$$r_{ij} = \frac{\sum_{b=1}^{N} d_{bi}\mu(d_{bi})}{\sum_{b=1}^{N} \mu(d_{bi})}; d_{bi} \in [0,10], \mu(d_{bi}) \in [0,1], i \in [1..n], j \in [1..m]$$

where d_{bi} represents the domain value (ranking) of service s_i for task t_j and $\mu(d_{bi})$ is the membership value for that domain point. Using this model, reputation ranking is more precise and trustworthy.

We showed above that non-functional quality criteria such as cost, time, availability and reputation, could be defined to better describe services. In the next sections, we will present web services composition and how these criteria can help in obtaining optimal compositions.

3. WEB SERVICES COMPOSITION

Web service composition originated from the necessity to achieve a predetermined goal that cannot be realized by a standalone service. Internally, in a composition, services can interact with each other to exchange parameters, for example a service's result could be another service's input parameter.

3.1 Problem Description

As an illustrative example, we will consider in this work a Travel problem. This scenario is a typical web services composition problem (Narayanam and McIlraith 2002) (OWL-S 2005). As far as creating the Travel service, we can use three atomic services (which are not composed) that will internally execute the travel; each one independently executes a task. A task can be described as an activity that applies to a specific domain. In this work, we treat activities and tasks identically. In our problem we will consider 3 tasks (BookFlight, BookHotel and RentCar) executed by 3 services (Airplane service, Hotel service and CarRental service). As explained in section 1 the planner will determine the execution order of these tasks. All the services resulting from the discovery process for a given task are candidate to execute this task. The aim of composition is to determine, out of all these candidate services, which one will belong to the composition.

3.2 Structure of Web Services Composition

The problem of composing web services can be characterized as a combinatory problem. As explained earlier, in the composition we have a set of services s_i, $i \in [1..n]$ that can execute a set of tasks t_j, $j \in [1..m]$. However, it is necessary to consider that one service can be dependent of other services. The main goal is to find the trade-off services composition, considering that there is a set of candidate services for each task.

In a composition, each service s_i is allocated to one task t_j. This association can be represented by a matrix (x_{ij}) where s_i represents the services and t_j represents the tasks. The matrix χ thus represents the services allocated to a composition.

$$\chi = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{pmatrix}$$

In our scenario the number of tasks and of services, m and n, are both limited to 3.

Actually, we can consider that a composition is a set of atomic web services or a set of composed web services. For instance, in the case of atomic services, if service s_1 is allocated to task t_2, it cannot be allocated to another task, because its domain is restricted to execution of task t_2. If we consider our Travel problem, a Hotel service cannot execute the bookFlight task, since it only deals with hotel reservations. On the other hand, considering that the composition may also have composed (non atomic) services, it means that one service can execute several tasks in the same composition. In our experimentations, we only consider atomic web services; this means that the sum of lines and that of columns in matrix χ should be 1.

$$\forall i \in [1..n], \forall j \in [1..m]$$
$$x_{ij} = \begin{cases} 1, \text{ if service } i \text{ is allocated to task } j \\ 0, \text{ otherwise} \end{cases}$$

The equation above determines whether a service belongs to a composition or not. It actually gives the result of our composition, since it defines, in the previous matrix whether service i is allocated to task j.

For instance, matrix χ' below represents one of the possible combinations in which service s_3 will execute task t_1, service s_1 will execute task t_2 and task t_3 will be executed by service s_2. As a result, this composition will be formed by services s_3, s_1 and s_2 respectively.

$$\chi' = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

An undetermined number of tasks, m, can be used to compose a service and an unlimited number of services, n, for each task t_j can be found. In fact, these possible combinations are considered for a predefined plan, which determines exactly in which order the tasks should be composed. However, concerning our architecture, the plan can also be changed, and so other possible combinations might be overseen. Moreover, if it is considered that p plans using m tasks can be created, the problem becomes even harder.

3.3 Models to Compose Web Services

The Web Service community is dealing with composition, interoperability between services, automated discovery and composition. Efforts have already been made by industrials and researches in order to achieve this goal. There are two main languages created in order to compose web services: BPEL4WS and OWL-S. Both languages are created focusing on activity-based models. In this way, BPEL4WS provides the basis for manually specifying composite web services. On the other hand, OWL-S is more ambitious and it provides a machine-readable description of web services which will enable automated discovery and composition (Hull and Su 2004). Indeed, there are other models to compose services such as: workflows, graphs, Petri nets and also currently programming languages as Java and C. Depending on each choice, composing web services can be harder and time consuming. Here we will focus on the two specific languages mentioned above: BPEL4WS and OWL-S. We will then illustrate some works using different models to compose web services.

3.3.1 Composing using BPEL4WS

Web services composition using BPEL4WS allows the manipulation of services as activities and processes. Actually, BPEL4WS language is a merge between Microsoft's XLang and IBM's WSFL, but all of them are considered as a web service flow language (van der Aalst 2003). As an executable process implementation language, the role of BPEL4WS is to define a new web service by composing a set of existing ones. The interface of the composite service is described as a collection of WSDL PortTypes.

A BPEL4WS process defines the roles involved in a composition as abstract processes. A buyer and a seller are examples of two roles. They are expressed using partner link definitions. We can have a role for each web service that is composed and does some activity. In order to integrate services, they are treated as partners that fill roles (Mandel and McIlraith 2003). BPEL4WS depends directly on the WSDL of the service. A business process defines how to coordinate the interactions between a process instance and its partners. Thus, a BPEL4WS process provides one or more WSDL services. The BPEL4WS process is defined only in an abstract manner, allowing only references to service portTypes in the partnerLink (Andrews et al. 2003). Each partner is characterized by a partner link and a role name. In summary, the main idea of business process is to create an organizer that point to each service endpoint that will be actually executed.

Characteristics. The distinction between roles and partners in a business process is an important characteristic of BPEL4WS. This allows more simple and intuitive integration between enterprises. Another important characteristic of BPEL4WS is the fault handlers. Faults handlers have the ability to catch errors in BPEL4WS. Another characteristic from BPEL4WS is message correlation that allows processes to participate in stateful conversations. It can be used to match returning or known customers to long-running business process. Furthermore, correlation mechanisms allow interaction between a service instance and a partner. BPEL4WS addresses correlations scenarios by providing a declarative mechanism to specify correlated groups of operations within a service instance (Andrews et al. 2002).

In a BPEL4WS process we define the interactions between these activities that compose the service. Thus, there are some types of interaction like sequence, flow, switch, pick, moreover, each one can be combined.

Implementation. We developed a prototype using BPEL4WS. We created our composition based on our simple Travel. Our composition has three services: Airplane, Hotel and CarRental. In BPEL4WS we define a composed service, such as Travel by describing which others services it contains. Figure 8-3, adapted from (Khalaf 2004), shows the relation between the Travel service and the others that compose it.

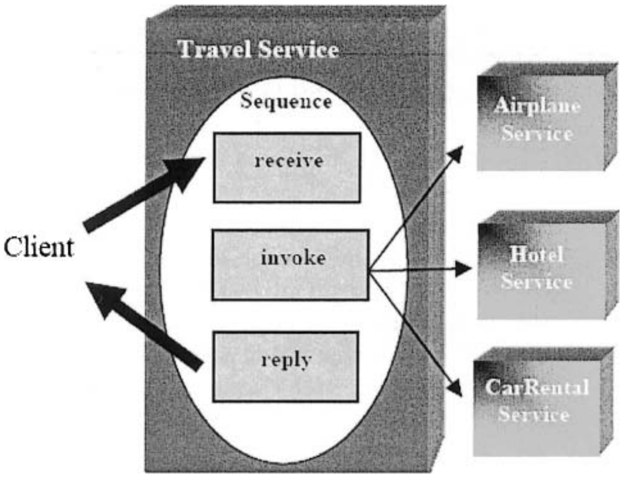


Figure 8-3. Internal view of Travel Service (BPEL4WS)

We put these three services in sequence, using the *sequence* structure. The *receive* structure indicates the location of the input variables in the sequence. The *invoke* structure is actually the service invocation. The *reply* is the response given by the sequence that here is the total cost of the travel. Between each structure, we can add an *assign* structure that is responsible for passing values between invoked services. See below our example using BPEL4WS:

```
<sequence name="TravelSequence">
     <receive partnerLink="client"
              portType="tns:travelPT"
              operation="trip"
              variable="request"
              createInstance="yes"/>
      <invoke name="invokeAirplane"
             partnerLink="airplane"
             portType="sairplane:Airplane"
             operation="bookAirplane"
             inputVariable="request"
             outputVariable="airplaneReturn">
      </invoke>
      <invoke name="invokeHotel"
             partnerLink="hotel"
             portType="shot:Hotel"
```

```
                    operation="bookHotel"
                    inputVariable="request"
                    outputVariable="hotelReturn">
            </invoke>
            <invoke name="invokeCar"
                    partnerLink="car"
                    portType="scar:Car"
                    operation="rentcar"
                    inputVariable="request"
                    outputVariable="totalReturn">
            </invoke>
            <reply partnerLink="client"
                    portType="tns:travelPT"
                    operation="trip"
                    variable="carReturn"/>
      </sequence>
```

After constructing the composition, we need to deploy our composite Travel service, making it available for execution. At this moment, the deployment engine will require the WSDL files that were related to partner's links. As we have an interaction with each service developed, we must have a WSDL for each one. We have to mention in each WSDL the grounding tag in order to actually find the service. Additionally, we invoke the composition using an API created by IBM called BPWS4J1.1 (BPWS4J 2004). Using this API to execute our composite service, we call a broker and we use the endpoint given by the Travel deployment to do the connection between the client and services' providers. Using the endpoint, the broker can find the service, and then it can pass the first parameters that are sent by the client.

3.3.2 Composing using OWL-S

The process of composing services using a semantic web language like OWL-S increases the automatic discovery and composition. In fact, OWL-S is based on ontology and OWL. This means that OWL-S is also based and constructed using resources and hierarchical concepts. With such a language, software agents can find services based on their computer-interpretable description.

The main motivating task for OWL-S was the ability to automatically discover web services. Other motivating tasks are automatic invocation of a service, with which a software agent can interpret markup to understand what input is necessary for the service call, what information will be returned and how to execute the service.

Additionally, the composed web service is actually an abstract service. In fact, the composition file has only the service calls. In OWL-S each service that is part of composition has the same structure as the composed one.

Characteristics. OWL-S is composed of three other structures called: service Profile, service Model and service Grounding, used to describe different aspects of the service (OWL-S 2005). The service Profile is responsible for presenting the service to other services or agents that want to use it. It describes the service in order to facilitate the search process, specifying what organization provides the service and what functions the service provides. See below a Profile example:

```
<profile:Profile rdf:ID="TravelProfile">
  <service:isPresentedBy
              rdf:resource="#TravelService"/>
     <profile:serviceName xml:lang="en"> Travel
     </profile:serviceName>
          <profile:textDescription xml:lang="en">
              Return travel: book flight, hotel, car rental.
          </profile:textDescription> ...
```

The service Model describes the service with regards to its inputs, outputs, effects and preconditions parameters. Furthermore, the process model is the core of OWL-S architecture; it defines how the process will be executed. Services can be composed using a combination of atomic or composite services. This implies that a composition can have services that are themselves composed. Additionally, in the service model we can say how the services will be executed: sequentially (*sequence*) or in parallel (*split/split+join*) or some other way (OWL-S 2005).

The service grounding is responsible for giving the endpoint of a service. A service grounding can be thought of as a mapping between an abstract and a concrete specification (OWL-S 2005). It is also in the grounding that we put the reference to each WSDL document.

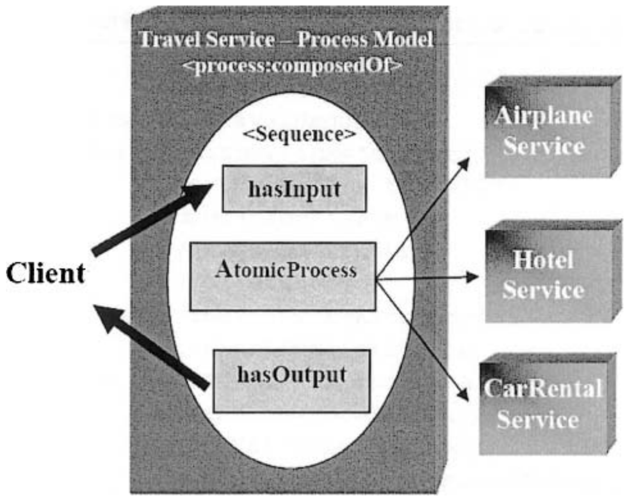


Figure 8-4. Internal view of Travel service (OWL-S)

Implementation. In our implementation using OWL-S composition, we defined the Travel service as being composed of three atomic services called Airplane, Hotel and CarRental services. We must define the OWL file for each atomic service. Furthermore, in these files we must put the grounding reference positioning exactly where the service is running. The Travel.owl file is only an abstract service where we define the input/output parameters and which service will be called. Figure 8-4 shows the internal view of Travel service.

After creating the OWL-S file containing the three services above, we can invoke the Travel service, sending it the parameters: *date_arrival*, *date_departure* and *destination_city*. As a result we will obtain the total amount for traveling. We also used a sequence structure in order to compose our services. In OWL-S we can pass values between services using *process:sameValues* structure.

```
<process:ProcessModel rdf:ID="TravelProcessModel">
  <service:describes
                  rdf:resource="#TravelService"/>
  <process:hasProcess
                    rdf:resource="#TravelProcess"/>
</process:ProcessModel>
<process:CompositeProcess rdf:ID="TravelProcess">
  <process:hasInput rdf:resource="#dt_arrival"/>
  <process:hasInput rdf:resource="#dt_departure"/>
  <process:hasInput
                rdf:resource="#destination_city"/>
  <process:hasOutput rdf:resource="#total"/>
  <process:composedOf>
    <process:Sequence>
      <process:components
                       rdf:parseType="Collection">
        <process:AtomicProcess
         rdf:about="Airplane.owl#AirplaneProcess"/>
        <process:AtomicProcess
               rdf:about="Hotel.owl#HotelProcess"/>
        <process:AtomicProcess
       rdf:about="CarRental.owl#CarRentalProcess"/>
      </process:components>
    </process:Sequence>
  </process:composedOf>
</process:CompositeProcess>
```

In order to execute the travel service, we have used OWL-S API (Mindswap 2004). For a client side, we defined an endpoint called Travel as the name of our service. Continue the execution, we invoke the Travel service and the OWL-S works on executing the others services that belongs to this composition.

It is important to highlight that these two examples were done in a statically way. In other words, we knew in advance which services would be part of the composition.

3.3.3 Other Web Service Composition Models

Many works opted for neither using BPEL4WS nor OWL-S. They modeled web services composition using other types of procedures.

In (Grigori and Bouzeghoub 2005) they propose modeling web services composition as graphs. In their work, even though they were worried about services match, the user requirements and the published service are graph based. The service retrieval approach is based on process graphs. Thus, a process is represented as a directed graph, whose nodes are activities. Edges have associated transition conditions expressing the control flow dependencies between activities.

In (Cardoso et al. 2004), they model web services composition using a workflow. In this work, a web service is considered as being a part of the workflow and it is argued that tasks and web services are treated with no difference. Between workflow and web services, both require tasks to have a structure which includes information such as task name, formal parameters, etc. Concerning web processes and workflows, in the authors' opinion, web processes can be viewed as workflows that manage web services instead of tasks. Thus, a workflow is composed of tasks and these tasks are actually web services.

In the work presented in (Narayanam and McIlraith 2002), web services compositions are modeled as Petri nets. In fact, all approaches mentioned above use graph representations. For instance, a Petri net is a bipartite graph containing places (drawn as circles) and transitions (drawn as rectangles).

Summarizing, several different manners exist for modeling web services composition; using various types of graphs, specific languages, etc.

4. PROBLEM MODEL

Many authors have studied the problem of web services composition, but only a few have worried about how complex this composition could be. Concerning our Travel problem, consider that we can now have more than ten tasks to be executed and over a hundred candidate services; with the daily growth of the Internet, these figures may soon be realistic. Thus, combining each task, respecting their restrictions and respectively finding the service to execute the tasks can be considered as a combinatory problem. Since we treat our services composition as a combinatory problem it requires optimization, so our Travel problem can be treated as an optimization problem.

Optimization problems require basically two elements: a search space composed of potential solutions and an objective function to be optimized. The search space may be restricted by a set of constraints. In our example, prior to execute the services, it is necessary to find optimal composition. In order to achieve optimal compositions we defined four main objectives that should be optimized: cost, time, reputation and availability. In addition to these objectives, we restricted the search space using constraints stating, for example, that one service can only be allocated to one task. Actually these objectives are our QoS model explained earlier. Since each QoS variable will be described inside a service, our optimization problem will retrieve these values in order to make possible combinations. The QoS (non-functional criteria) model was used as the objectives to be optimized because we need to differentiate candidate services with identical functionalities. In the next subsections we explain our objectives and the constraints we used in detail.

4.1 Objectives

Our problem consists of four objectives. The first one is cost minimization:

$$Min\sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}\, p_{ij}\, x_{ij}$$

In this problem, c_{ij} represents the cost criterion in the quality model. It defines the cost of using service s_i for executing task t_j. p_{ij} indicates the service's ability to execute a given task. Since we can have atomic or composed services belonging to the composition, not all of the discovered services will be able to execute all the tasks. Thus, p_{ij} is a binary variable informing whether service s_i is able to execute a task t_j or not. The binary variable x_{ij} is responsible for expressing if a service belongs or not to the composition. This is represented in matrix χ.

Another objective concerns time. As explained in the QoS model, time is the elapsed time between the request and the response. The time objective also needs to be minimized:

$$Min\sum_{i=1}^{n}\sum_{j=1}^{m} t_{ij}\, p_{ij}\, x_{ij}$$

In our model, t_{ij} concerns the time taken by service s_i to execute task t_j. The other variables p_{ij} and x_{ij} are those explained above.

The availability objective shows the probability that a service can be accessed and used. In our case, it should be maximized, because it is preferable that this probability is as high as possible.

$$Max\sum_{i=1}^{n}\sum_{j=1}^{m} a_{ij}\, p_{ij}\, x_{ij}$$

Variable a_{ij} should belong to [0,1].

The last objective is related to the reputation a service has in a determined field.

$$Max\sum_{i=1}^{n}\sum_{j=1}^{m} r_{ij}\, p_{ij}\, x_{ij}$$

r_{ij} stands for the reputation service s_i has when executing task t_j. This objective needs to be maximized because the higher the reputation the better the service is judged.

Using our objectives, we can now reconsider our Travel problem. Cost represents the price of a service execution and Time is the execution time of a service. Moreover, Availability is the probability a service is "alive" and

Reputation is the trustworthiness of the service in a determined field. We can easily understand that some clients do not give any preference to cost and prefer spending more money on travel, provided it is on a reliable airline company. In fact, we want to consider the four objectives simultaneously for travel.

In fact, even if the four objectives are contradictory with each other, we do not give any preference to any one of them. This means that we do not need to give them a weight. For instance, we do not want to give any preference to cost over time. Thus, the service with the smallest cost will not necessarily be part of our composition, since its other measures of quality must be considered. We will explain how one can treat this kind of problem in section 5.

4.2 Constraints

In our model the solutions of our problem must also satisfy two constraints. The first one states that only one service in a composition is allocated to each task. It can be represented by:

$$\sum_{i=1}^{n} x_{ij} p_{ij} = 1, \forall j \in [1..m], x_{ij} \in \{0,1\}$$

where x_{ij} specifies whether or not a service belongs to a composition. Variable p_{ij} represents the capacity of service s_i to execute task t_j. Thus, this first constraint specifies that each task in the composition must be executed by exactly one service.

The second constraint concerns the user's budget.

$$\sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} x_{ij} \leq W, W > 0, x_{ij} \in \{0,1\}$$

This constraint states that the cost of using the resulting composition should not exceed a given value W.

5. MULTIOBJECTIVE OPTIMIZATION

As explained in section 4 we have four objectives that we want to minimize and maximize. However, neither a preference nor a weight should be given to any one of them. We want to treat all of them together and simultaneously. Although single-objective optimization problems may have a unique optimal solution, Multiobjective Optimization Problems (MOP) present a possibly uncountable set of solutions, which when evaluated, produce vectors whose components represent tradeoffs in objective space. A decision maker then implicitly chooses an acceptable solution by selecting one or more of these vectors (Coello et al. 2002;Tan et al. 2005;Deb 2001;Collette and Siarry 2003).

Multiobjective optimization allows the co-existence between two or more objectives that are normally contradictory. Two objectives are contradictory if the decrease of one of them implies the increase of the other. Another important feature is that in a multiobjective problem we do not have only one optimal solution but a set of solutions. These solutions are called *Pareto solutions* (Tan et al. 2005).

Thus, MOP can be defined as finding (Osyczka 1985): "a vector of decision variables which satisfies constraints and optimizes a vector of function whose elements represent the objective functions." This is formally defined in (Coello et al. 2002) as:

Find the vector $\vec{x} = [x_1, x_2, \ldots, x_n]^T$ which satisfies the m inequality constraints:

$g_i(\vec{x}) \geq 0,\ i = 1,2,\ldots,m$

and optimize the vector function

$\vec{f}(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x}), \ldots, f_k(\vec{x})]^T$

The constraints define the feasible region and any point in $\vec{x}$ defines a feasible solution. T stands for vector transposition. Thus, the points inside the feasible region satisfy all defined constraints.

A large number of approaches exist to resolve multiobjective optimization problems. Some of them use the knowledge they have about the problem to give preferences to some objectives, thus bypassing the multiobjective aspect. Others give all objectives the same level of importance, etc. Among these approaches, we should distinguish between two categories: non-Pareto and Pareto approaches. Non-Pareto approaches do not actually treat the problem as a multiobjective problem. They try to convert it into a mono-objective problem. On the other hand, Pareto approaches do not transform the problem's objectives, but try to optimize them simultaneously.

5.1 Non-Pareto Approach

There are many non-Pareto approaches; however, we focus here on two of them used in multiobjective problems.

5.1.1 Objective aggregation method

This method is the most commonly used in multiobjective optimization problems. The goal is to transform the multiobjective problem into a mono-objective problem. Hence, they use a weight mechanism to aggregate all objectives into a unique objective. This approach has the advantage of being able to reuse all classic algorithms used for solving mono-objective optimization problems. However, the weights must be given with attention because it impacts directly into the solutions.

5.1.2 ε-Constraint

This is another manner of transforming a multiobjective problem into a mono-objective one. When confronted with a problem consisting of *m* objectives, we convert *m-1* of them into constraints. Thus, the idea is to optimize the preferred objective, considering all the others as constraints. This method is also known as the trade-off method.

5.2 Pareto Approach

Having several objective functions, the notion of "optimum" changes, because in MOP, the aim is to find good compromises ("tradeoffs") rather than a single solution. We can say that $\vec{x}$ is Pareto optimal if there exists no feasible vector $\vec{y}$ which decreases some criterion without causing a simultaneous increase in at least one other criterion (Coello et al. 2002).

5.2.1 The Relation of Dominance

Despite the fact that we have obtained many solutions resolving our multiobjective problem, only a restricted number of them will actually be relevant. Thus, in multiobjective problems, in order to consider an interesting solution, we need to have a means of determining the most relevant solutions. In order to determine these solutions, a relation of dominance is defined as follows:

Definition: The relation of dominance in a minimization problem is defined in (Coello et al. 2002) as:

Vector $\vec{v}$ dominates vector $\vec{\tau}$ ($\vec{v} \preceq \vec{\tau}$) if, and only if:

$\vec{v}$ *is partially less than* $\vec{\tau}$

i.e. $\forall i \in \{1,\ldots,k\}, v_i \leq \tau_i \wedge \exists i \in \{1,\ldots,k\}: v_i < \tau_i$

Solutions that dominate other solutions but which do not dominate each other are called optimal solutions in the sense of Pareto (or nondominated solution).

5.2.2 MultiObjective Evolutionary Algorithms

The use of Evolutionary Algorithms (EA) to solve Multiobjective problems has been motivated mainly because of the population-based nature of EAs which allows the generation of several elements of the Pareto optimal set in a single run. The Multiobjective Evolutionary Algorithms (MOEA) are among the most powerful resolution methods for multiobjective optimization (Coello et al. 2002). MOEA take into account contradictory objectives and allow finding a set of nondominated solutions. An evolutionary algorithm is composed of three fundamental elements:

- Population: it is composed of individuals that represent potential solutions
- Evaluation: it is a mechanism that allows individual evaluations in order to measure the individual adaptation into an environment.
- Evolution: it is the mechanism that allows the population evolution. Evolution is ensured by selection, crossover and mutation.

The selection mechanism determines the individuals that can reproduce its characteristics in future generations. The crossover is the mechanism responsible to create new individuals based on parents' characteristics. The mutation mechanism introduces limited changes in the individuals.

Genetic Algorithm to MOP (NSGA-II). The NSGA-II (Nondominated Sorting Genetic Algorithm) (Deb el al. 2002) used in this work is one variation of Goldberg's Pareto ranking (Goldberg 1989), though any other MOEA such as SPEA(Zitizler and Thiele 1998), PAES (Knowles and Corne 1999) and PICPA (Barichard and Hao 2003) could have been used.

In NSGA-II, the tournament selection, crossover and mutation operators are used to create a child population that will be added to a result population given by the later generation. The new population is sorted based on non-domination. In this step, elitism is ensured because the best nondominated sets will be chosen for the next population. Using constraints, the relation of domination between two individuals can be characterized as a feasible or unfeasible solution. Thus, the ranking will be done based also on feasible solutions.

Applying NSGA-II to our Travel problem, a chromosome corresponds to a services composition which is defined by a 0/1 string. Each binary variable that represents a gene indicates whether the service belongs to the optimal composition or not. The example below shows a chromosome representing a solution of a services composition problem using 15 services and 3 tasks (each of them having 5 candidate services):

0	0	0	1	0	1	0	0	0	0	0	0	1	0	0
genes: 1-5					genes: 6-10					genes: 11-15				

Each service is represented in the above chromosome by a binary variable (a gene) and the binary variables (genes) are grouped according to the task they are candidate for (genes 1-5: task 1, genes 6-10: task 2, genes 11-15: task 3). For each group of 5 binary variables, only one service will belong to our composition. This chromosome corresponds exactly to our matrix χ and means that service s_4 is allocated to task t_1. Task t_2 will be executed by service s_6 and task t_3 by service s_{13}.

6. RELATED WORK

Many authors have proposed quality of service models for selecting web services. Some authors applied their QoS model to agents based architectures, others to centralized registries or to individual services.

In (Ran 2003) the main idea is to include a QoS model into UDDI registries so that QoS parameters can be included as search criteria. In fact, they propose to use a QoS model as non-functional requirements to enable a service search based on functional and non-functional (QoS) parameters. They also explain that the current UDDI model limits the service discovery to functional requirements. Due to this limitation, they propose to incorporate a QoS model into UDDI registries. The proposed model will coexist with the current UDDI. If no services are found with these qualities, feed-back is returned to clients and so they can reduce their quality values.

In (Sreenath and Singh 2004) the authors propose a mutual evaluation process between agents to select a web service. It selects the best service based on rates given to providers by agents. A provider is ranked by an agent and the agent's evaluations are, themselves, evaluated by other agents. Thus, selecting a service provider involves getting a list of rated service providers and choosing the best based on a weighted average calculation. The result of the execution of the chosen service is then feedback into the service provider rating mechanism.

The main idea in (Cardoso et al. 2004) is an adaptation of Workflow Quality of Services and its transposition to web service technologies. First of all, they propose to characterize workflows based on their QoS in order to better fulfill customers' expectations. The QoS model is composed of: time, cost, fidelity and reliability. Fidelity means how well workflows, instances and tasks are meeting user specifications. Concerning reliability, it is the measure of the likelihood that the component performs a task demanded by a user. These QoS constraints are implemented into METEOR workflow management systems for Genomic Projects.

Ideas in (Zeng et al. 2003) are very close to our proposition regarding the QoS model and also to the resolution method. This work treats the services selection during the execution process and so it takes into account multiple criteria. Thus, the idea is that services are selected by the composite service execution engine based on a set of criteria. This paper presents a quality model that is characterized by non-functional properties: price, duration, reputation and availability. Service selection is then formulated as an optimization problem and a linear programming method is used to compute optimal services execution plans to compose services. This work is an example of objective aggregation approach. In other words, they weight the objectives and then sum them all in order to create a single aggregate objective. The transformed problem is solved using linear programming. Notice that this approach cannot lead to alternative solutions and is not able to handle automatically non-linear constraints. The most important difference between our work and Zeng et al's work (Zeng et al. 2003) is that, as opposed to their work, we do not give any weight to any objective. We treat all objectives with the same importance using a multiobjective optimization approach. Even though our objectives are contradictory, they are taken into account simultaneously by our resolution algorithm.

In (Liu et al. 2004), in order to improve the work of (Zeng et al. 2003), the authors propose specific domain criteria for each service that will be selected. Thus, QoS information is collected from the properties of services as they are published by providers. The main idea is that some users want to select services based on time while others only want to consider cost. Thus this paper proposes a QoS model based on user preferences.

In (Canfora et al. 2005), the authors propose a QoS-aware composition based on run-time values. They argue that QoS values based on estimation may differ from those at runtime. Thus they prefer to use runtime QoS value when composing services in order not to go against SLA accords. An example is that, at runtime, some services may not be available when, according to estimations, they should be. Thus, this framework needs to reconsider services composition in order to change the bindings between abstract and concrete services.

Ideas in (Jaeger et al. 2005) discuss how the selection can consider different QoS categories to determine the most suitable candidates for the composition. If more than one category is used for optimization, a multi-dimensional optimization problem arises. On the other hand, if exactly one category is relevant, an algorithm chooses the candidate that offers the optimal value. For each task the candidate that offers the best QoS constraint category is assigned. Thus, if a combination which respects the constraints exists, it is found.

In (Bonatti and Festa 2005) the authors consider optimal services selection based on a given set of service requests (i.e. activities occurring in a workflow), a set of available services (offered services), result of the matchmaking process (association of the request and the offer) and a numeric preference measure. Their selection is based on cost and two different QoS-like criteria. These criteria are ordered and static.

7. CASE STUDY

One of the main contributions of this work concerns the multiobjective optimization approach. As explained earlier, we consider that objectives and solutions should be searched considering these four criteria simultaneously. To achieve this, we use the multiobjective evolutionary algorithm NSGA-II. The next sections describe our experimentation using the NSGA-II for composing web services.

7.1 Experimentation

Applying this algorithm to our problem, several experiments using our composition model were done in order to find optimal compositions.

7.1.1 Tests set

The main objective of our tests was to find a set of Pareto optimal compositions from which a user can select her preferred solution. The first test that we did was to analyze the same number of services and tasks, changing the number of generations and populations. The number of services was set to 30 and the number of tasks to 3. We chose to allocate the same number of candidate services to each task. The aim of this experimentation was to analyze how the algorithm treats services composition.

The next test that we did was aimed at studying the scalability of the services composition algorithm with respects to the number of candidate services and to the number of tasks. Population and generation were kept constant in all experiments, but the number of services and tasks was changed. In fact, we increased candidate services for each tasks. The population was fixed to 200 individuals and the generations were fixed to 500. These values were taken considering other experiments using the NSGA-II algorithm.

As for the previous experiment, we also consider that the numbers of candidate services for each task are equal. The number of services is thus equal to the number of variables, because each service is represented as a variable in our model.

7.1.2 Algorithm Parameters

In the first experiment we used population ranges from 10 to 200 and generation ranges from 10 to 500. The crossover probability was 0.9 and the mutation was $1/l$ where l is the number of binary variables. In our case, we used 30 binary variables because we have 30 services. These 30 binary variables represent 3 tasks and each task can be executed by 10 candidate services. The crossover used was single-point. We used 4 objective functions and 2 constraints as previously defined in our model. The first constraint determines the candidate services and the other one represents the maximal budget given by the user. This value was fixed for all compositions. The QoS values were given randomly to each service.

In the second test, the population size was set to 200 and generation to 500. We did these experiments using 30 and 60 services with 3 and 5 tasks. It means that, for example, using 60 services and 3 tasks, we have 20 candidate services equally distributed for each task. The crossover mutation and probability was maintained (of course they changed according to the number of variables). In both experiments, all constraints must be satisfied in all generations and thus only feasible solutions were selected for the next generation.

7.1.3 Results

The results of our experiments consist of a set of chromosomes; each one representing a services composition. Since we defined a population size of 200, the maximum number of solutions found was also 200. However, out of these solutions we only highlighted the distinct Pareto optimal solutions.

In Figure 8-5, we show the evolution of our model based on the number of distinct Pareto optimal solutions found for 30 services and 3 tasks. We can see that 70 distinct solutions are found for a population size of 200 and a generation size of 500. The tradeoff solutions do not violate any constraints.

Using 30 services for 3 tasks, the algorithm gives 70 distinct nondominated solutions in approximately 18 seconds.

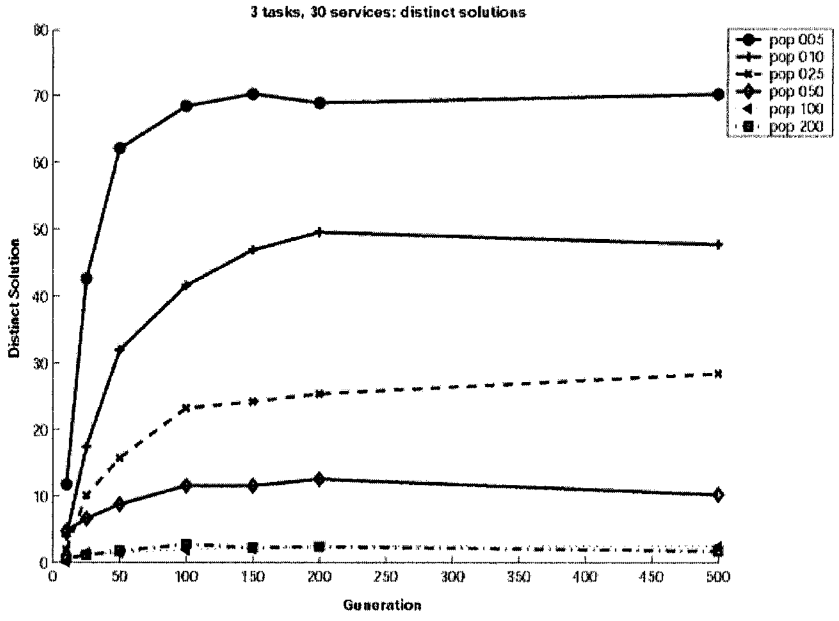


Figure 8-5. Distinct Pareto solutions

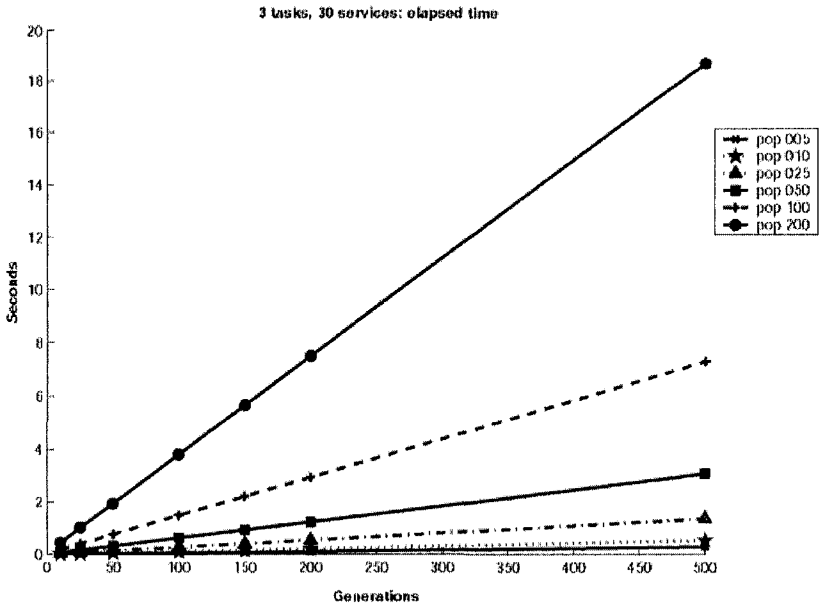
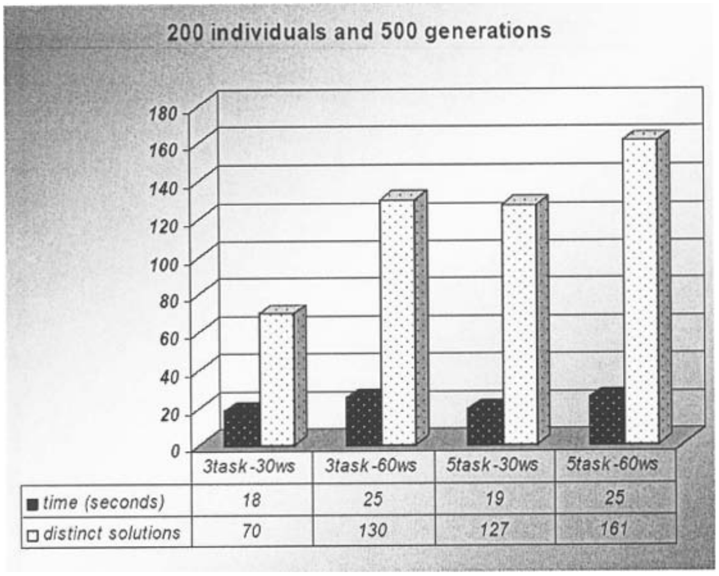


Figure 8-6. Elapsed Time

We notice, in Figure 8-6, that it is not necessary to use large Distinct Pareto solutions populations since for a population size of 100, the 47 distinct solutions are obtained in 7 seconds.

The next experiment consisted in changing the number of services and the number of tasks. In Figure 8-6 we observe that as the number of services increases, more solutions are found. In addition, as the number of candidate services increases, the elapsed time to find the solutions also increases.

For example, using 60 services for 3 tasks means that there are 20 candidate services. However using 60 services for 5 tasks, there are only 12 candidate services. The difficulty in finding tradeoff solutions increases with the number of candidate services. Augmenting the number of tasks also means increasing the number of constraints and so facilitating the achievement of Pareto optimal compositions, as shown in Figure 8-7.



	3task-30ws	3task-60ws	5task-30ws	5task-60ws
■ time (seconds)	18	25	19	25
□ distinct solutions	70	130	127	161

Figure 8-7. Services and Tasks

8. CONCLUSIONS

In this paper we have explained how services could be selected in order to make optimized compositions. We proposed some improvement on quality models, highlighting the reputation criterion. We based the calculation of reputation on fuzzy numbers. Using non-functional features (QoS) for the optimization of composite services may lead to contradictory objectives. However, we do not wish to give any preference (weight) to any of these objectives. Thus we chose to treat services composition as a multiobjective problem. We used the multiobjective evolutionary algorithm called NSGA-II and obtained a set of optimized compositions representing different tradeoffs. The experimentations carried out validate our approach and show its feasibility in solving the Travel problem.

9. QUESTIONS FOR DISCUSSION

Beginner:

1. Why do we need to compose web services?

2. What is the difference between static composition and automatic composition?
3. List different techniques used for composing automatically web services.

Intermediate:
1. Should QoS values be assigned to web services or should they be associated to service providers?
2. List other possible approaches to solve the multiobjective model for the optimization of web services composition?
3. Could the availability criterion be a continuous measure? Why?
4. Why is it necessary to optimize the composition?

Advanced:
1. In our problem, what happens if the number of services and tasks is increased?
2. What are the benefits of using multiobjective approaches?

Practical Exercises:
1. Choose an example to compose statically using three services. Develop it using OWL-S or BPEL4WS.
2. Take a composition example, enumerate all possible compositions, choose a quality criterion and try to optimize using a linear programming approach.

10. SUGGESTED ADDITIONAL READING

- Coello Carlos A., van Veldhuizen D.A., Lamont G.B.; *Evolutionary Algorithms for Solving Multi-Objective Problems.* Kluwer Academic/Plenum Publishers, New York, 2002: This book is a reference in the domain of Evolutionary Multiobjective Optimization.

11. ACKNOWLEDGMENT

Daniela Barreiro Claro is supported by a research scholarship given by the Région du Pays de La Loire (2003-2006).

12. REFERENCES

Andrews T., Curbera F., Dholakia H., Goland Y., Klein J., Leymann F., Liu K., Roller D., Smith D., Thatte S., Trickovic I. and Weerawarana S. Specification: BPEL4WS - Business Process Execution Language for Web Services - Version 1.1. Retrieved May 30, 2005, from ftp://www6.software.ibm.com/software/ developer/library/ws-bpel.pdf, May (2003).

Barichard H., Hao J-K. A population and Interval Constraint Propagation Algorithm. In Second International Conference Evolutionary Multi-Criterion Optimization (EMO). Lecture Notes in Computer Science 2632:88-101(2003).

Bonatti P., Festa P. On Optimal Service Selection. In International World Wide Web Conference (WWW'2005), May 10-14, Chiba, Japan (2005).

BPWS4J API. Retrieved November 26, 2004, from http://www.alphaworks.ibm.com/ tech/bpws4j.

Canfora G., di Penta M., Esposito R., Villani M.L. QoS-Aware Replanning of Composite Web Services. In International Conference of Web Services (ICWS'2005), July 11-17, Orlando (2005).

Cardoso J., Sheth A., Miller J., Arnold J., Kochut K. Quality of Service for Workflows and Web Service Processes. In Web Semantics: Sciences, Services and Agents on the World Wide Web 281-308, 1 (2004).

Collette Y., Siarry P. Multiobjective Optimization: Principles and Case Studies, Springer, NY, Berlin (2003).

Coello C.C.A., Van Veldhuizen D.A, Lamont G.B. Evolutionary Algorithms for Solving Multi-objective Problems. Kluwer Academic Publishers, New York (2002).

Deb K. Multi-Objective Optimization using Evolutionary Algorithms. John Wiley \& Sons, ISBN 0-471-87339-X, Chichester, UK (2001).

Deb K., Pratap A., Agarwal S., Meyarivan T. A Fast and Elitist Multi-Objective Genetic Algorithm: NSGA-II. IEEE Trans Evol Computat, Volume 6, pp. 182-197, April, (2002).

Fuzzy Logic Fundamentals, Chapter 3, pg 61-103. Retrieved February 8, 2005. Available on http://www.informit.com/content/images/0135705991/ samplechapter/0135705991.pdf (2005).

Goldberg D.E. Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Publishing Company, Reading, Massachusetts (1989).

Grigori D., Bouzeghoub M. Service retrieval based on behavioral specification. In International Conference of Web Services (ICWS'05), July 11-17, Orlando (2005).

Hull R., Su J. Tools for Design of Composite Web Services. In SIGMOD 2004, June 13-18, Paris (2004).

Jaeger M.C., Mühl G., Golze S. QoS-aware Composition of Web Services: A Look at Selection Algorithms. In International Conference of Web Services (ICWS'2005), July 11-17, Orlando (2005).

Khalaf R. Business Process with BPEL4WS, Part 2. Retrieved October 27, 2004. Available on http://www-128.ibm.com/developerworks/webservices/library/ws-bpelcol2/

Knowles J., Corne D. The Pareto archived evolution strategy: A new baseline algorithm for multiobjective optimization. In Congress of Evolutionary Computation, Piscataway, New Jersey: IEEE Service Center, 98-105 (1999)

Liu Y., Ngu A.H.H., Zeng L. QoS Computation and Policing in Dynamic Web Service. In Thirteenth International Conference of WWW 2004, May 17-22, New York, New York (2004).

Löstedt J., Svensson M. Baltazar - A Fuzzy Expert for Driving Situation Detection. Master Diss., Department of Sciences, Lund University (2000).

Mandel D.J., McIlraith S.A. Adapting BPEL4WS for the Semantic Web Bottom-up Approach to Web Services Interoperation. In Second International Semantic Web Conference (ISWC), Sanibel Island, Florida (2003).

Mindswap G. Maryland Information and Network dynamics lab semantic web agents projects. Retrieved October 28, 2004. Available on http://www.mindswap.org/ 2004/owl-s/api/index.shtml (2004).

Moura L. A Genetic algorithm to fuzzy multiobjective optimization. Master diss. Department of Electric Engineer, Campinas University (2001).

Narayanan S., McIlraith S.A. Simulation, Verification and Automated Composition of Web Services. In Eleventh International World Wide Web Conference (WWW 2002), Honolulu, May 7-10 (2002).

Osyczka A. Multicriteria optimization for engineering design. In Gero, J.S., editor Design Optimization, pg.193-227. Academic Press (1985).

OWL-S Coalition. OWL-S: Semantic Markup for Web Services. Retrieved April 12, 2005. Available on http://www.daml.org/services/owl-s/1.1/ (2005).

Ran S. A Model for Web Services Discovery with QoS. In ACM SIGecom Exchanges, Volume 4, Issue 1, Spring, pp. 1-10, ACM Press, New York, NY (2003)

Sreenath R.M., Singh M.P. Agent-based service selection. In Web Semantics: Science, Service and Agents on the World Wide Web, 261-279 (2004).

Tan K.C., Khor E.F., Lee T.H. Multiobjective Evolutionary Algorithms and Applications. Springer-Verlag, ISBN 1-85233-836-9, London (2005).

van der Aalst W.M.P. Don't Go with the Flow: Web Services Composition Standards Exposed. IEEE Inteligent Systems, 18(1):72-76 (2003).

Zeng L., Benatallah B., Dumas M., Kalagnanam J., Sheng Q.Z. Quality Driven Web Services Composition. In Twelfth International Conference of WWW, May 20-24, Budapest (2003).

Zitizler E., Thiele L. Multiobjective Optimization using Evolutionary Algorithms - A Comparative Case Study. Parallel Problem Solving from Nature V, A.E.Eiben, T.Bäck, M.Schoenauer and H-P. Schwefel Eds. Berlin, Germany: Springer, 292-301 (1998).