RA221 (W0411-084) November 10, 2004
Conputer Science

IBM Research Report

ICSOC 2004 Proceedings - Short Papers

Sanjiva Weerawarana (editor)
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Automatic Configuration of Services for Security, Bandwidth, Throughput, and Availability

Garret Swart*
University College Cork
Cork, Ireland
g.swart@cs.ucc.ie

Benjamin Aziz
University College Cork
Cork, Ireland
b.aziz@cs.ucc.ie

Simon Foley
University College Cork
Cork, Ireland
s.foley@cs.ucc.ie

John Herbert
University College Cork
Cork, Ireland
j.herbert@cs.ucc.ie

ABSTRACT
The process of efficiently deploying a complex system of services on a complex network of servers is tedious and error prone, with many properties to check and many possibilities to examine. Automated tools are needed to turn this into a humanly tractable problem. We present a precise model of a service-oriented computing system that allows many important configuration properties to be defined and optimized for, including throughput, network bandwidth, security and availability. We transform this model into a system of constraints that can then be solved using mathematical and constraint programming yielding an optimal system configuration that meets all the stated requirements. We have implemented this in OPL and have used it to generate optimal configurations for realistic systems with tens of services running on hundreds of servers communicating on multiple subnets.

Categories and Subject Descriptors
C.2.0 [Computer Communication Networks]: General – Security and protection. C.2.3: Network Operations – Network management, C.4.0: [Performance of Systems] Modeling techniques, Performance attributes K.6.4: [Management of Computing and Information Systems] System Management – Quality assurance

General Terms
Management, Performance, Security.

Keywords
Automated Server Provisioning, Service-oriented Computation, Automated Network Management, Autonomous Computing, Quality of Service (QoS)

1. INTRODUCTION
The process of deploying complex systems of services on a complex network of servers is fraught with peril for the systems administrator. No one wants to be on the front page of the New York Times when a multi-million dollar system becomes inaccessible due to an unforeseen problem with an obscure configuration setting. No one wants to be asked by the boss why the network has to be upgraded when a simple reconfiguration might be adequate. No one wants to spend days looking at ways of making do with the current hardware because the budget is used up but new applications are coming online. Good systems configuration needs precise checking and exhaustive optimization, not just rules of thumb and heartfelt prayer.

We suggest solving this problem by the use of mathematical modeling. Methods rooted in mathematics can result in configurations that are provably optimal and correct. Methods rooted in mathematics can be amenable to machine reasoning rather than the more onerous and error prone human variety. Using formal techniques we can translate engineering intuition into mathematical constraints.

In this paper we define a precise model of a service-oriented computing system. We argue that this model is close enough to reality to be interesting. We then define various properties of the model that correspond to important properties in real systems. The properties that we define and optimize for are network security, server throughput, service availability and network bandwidth. All of these terms have been bandied about enough that they need careful definitions, and we define them using our mathematical model.

We then encode this model and properties into an Optimization Programming Language (OPL) application so that a combination of mathematical and constraint programming techniques that are part of the OPL implementation can be brought to bear on this problem to produce a set of optimal assignments of logical components to physical resources. Using the facilities of OPL, we write a system model that defines the data to be presented and its constraints. This model can be instantiated to represent any computing system that falls within the model. Once instantiated, the model can be solved by OPL to find the optimal configuration of resources that meets the requirements. This separation of the model, its instantiation and solution technique allow such systems to be used by systems administrators without a degree in operations research.

Finally we show the results of this model when applied to a realistic system containing 26 services and 240 servers on 5 subnets.

Novel aspects of this work include:

– The application of configuration modeling and optimization to general service oriented computation. The service model that we introduce and the complex relations that services can have with each other allows us to model service-oriented systems “from the phosphor to the oxide.”

– The simultaneous modeling of many configuration properties so that the values of these properties can be played off against each other, and a framework that allows even more properties to be defined and modeled.

*This author’s current address is: IBM Almaden Research Center; 650 Harry Road; San Jose, California 95120 gswart@almaden.ibm.org.

– The careful definition of quantifiable security properties that correspond to properties that security experts attempt to optimize for. Security is often thought of as a binary property but the use of security metrics allows greater flexibility to the configuration process.

Modeling and optimization, while at the core of automated configuration, is not all that is needed. Other aspects of this problem include:

– Model reverse engineering: Generating a model from a real system is a huge task. System administrators need tools to facilitate generation of system requirements from a real system that is known to meet its requirements. The resulting model will allow the search for cheaper configurations meeting the same requirements or new configurations meeting adjusted requirement, E.g. Build me a system that behaves just like my old system but that handles twice the load at half the cost.

– Requirements understanding: A complex system may have thousands of requirements. How can we ensure that all the important requirements for the correct working of a system have been captured? Missing a requirement may produce a system that, while it meets all of the stated requirements, does not function. The implications of security requirements are notoriously difficult to understand but we want to ensure that whatever language is used for these requirements supports the WISIWIM requirement: "What I Said Is What I Meant."

– Configuration deployment: Once a new configuration has been determined, how can we reliably implement the change from the current configuration to the new configuration? This may involve rewiring network connections, reconfiguring routers, redeploying services on different servers. Each task should be automated if possible, and in any case checked for correct completion.

– Model refinement: Any abstract model is just that, a model. It is meant to mirror reality and any place where the model does not mirror reality should be flagged and fixed. For example, the model may predict that adding a processor in a particular role will improve performance by 20%. If deploying the change unexpectedly results in a performance improvement of 10% or 50%, we want to adjust our model so that properties of future configurations will be more accurately predicted. When many changes are made simultaneously figuring out where the model should be changed can be difficult.

– Adaptability: A static service configuration is unlikely to stay unchanged for long. New services are being introduced and the properties of existing services change based on market success, developing usage patterns, and service implementation changes. We want to find an optimal sequence of configurations from the current optimum, based on one set of assumptions and requirements, to a new optimum based on a new set of assumptions and requirements.

In this paper we focus on the necessary first step: the modeling and searching for static configurations; other aspects are left to future work.

Service configuration modeling is important to designers of services because it can impact the way they think of their services and the information they need to specify about them. It is important to vendors of service software because it allows complex networks of service software to be installed by the users themselves, rather than by teams of warring vendor consultants. It is important to service-oriented middleware providers, as they will need to provide the tools that provide the facilities listed above. Finally, it is important to those who deploy services, as they will be able to save money and avoid worry as they deploy their service networks.

2. MODELING A SERVICE-ORIENTED SYSTEM

The art of modeling lies in figuring out what to put in and what to leave out. Putting too much in the model results in a model that can be intractable for both humans and machines, while leaving too much out of the model makes important questions impossible to state. The goal of this paper is to leave enough in the model to be able to do realistic network capacity planning, server capacity planning, network security planning and service level availability planning. Anything that is not required to meet those requirements, we left out. In this section we define the components of our model and the information a user of the system has to provide about each component and the information that the optimizer will produce. In the next section we describe how we use this information to define properties of the system that meet the planning needs of system administrators. A UML class diagram showing the relationship between the components is shown in Fig. 1.

Service. The fundamental system component in a service-oriented architecture is, of course, the service. In this context we define a service to be an entity that can perform a set of operations on behalf of callers on a defined set of data. For example, Hertz may offer a car rental booking service that allows clients to book its cars. Avis may offer a distinct service that provides access to its cars. Travelocity and LastMinute may each run a travel agency that offers services that allow clients to book cars on either Hertz or Avis. These form four distinct services.

Implicitly associated with a service is that service's implementation. This implementation, while invisible to the user of the service will determine certain properties of the service, e.g. the load caused by performing an operation or the set of services called by this service.

We denote the set of all services being modeled as a set named *Services*.

To specify the load generated by an invocation of a service, each service must be provided with a per invocation load amount that must be provided by the servers assigned to run this service. This is the expected amount of load caused by handling a single call on the service. The units of this cost might be Java Virtual Machine instructions or any other reasonable unit of execution cost that adequately represents the utilization of resources on a machine running the service. As processor designers, compiler writers, or even marketing directors will tell you, there is no one number that can represent a server's capacity or an application's load, but for this purpose we'll assume that we have one that provides an adequate estimate.

Formally we define a function *loadU* that defines the expected load units caused by a single invocation of the service.

$$loadU : Service \rightarrow \Re^{+}$$

Service Interface. Each service implements a certain protocol or language to facilitate communication with it and other similar services. We call this protocol the interface to the service. To facilitate interoperability, many services may implement the same interfaces. In a Web Services infrastructure an interface may be specified as a WSDL object and identified by a URL. In a Corba infrastructure, an interface may be specified by an IDL file and identified by a UUID.

Formally we can represent this as a set *ServiceInterface* and an *implements* function that represents the relationship between the service interface and the services that implement it. Note that each service implements only a single interface; we will see below that the concept of a deployable unit, which represents an aggregation of services, provides for the functionality of allowing multiple interfaces to be implemented by a single service.

$$implements : Service \rightarrow ServiceInterface\ .$$

We do not consider the process of how service interfaces may be mapped to services using other services like LDAP or UDDI or how interfaces might be discovered using an ontology.

Service dependencies. Services may be composed from other services. For each service we assume we have a set of services that are used in this service's implementation and that we have determined the expected number of invocations of those subsidiary services for each invocation of the entry service. This can sometimes be determined by code inspection and sometimes by measurement. Even if service binding is done dynamically, data can be collected on the long-term behavior of a particular installation.

For example, Travelocity's car rental service may invoke the Hertz and Avis services an average of 1.4 times per invocation, while LastMinute's car rental service may invoke the Hertz service 1.7 times and the Avis service 2.2 times per call.

As in this example, each subsidiary service may be used by any number of layered service implementations. For simplicity, we assume that there are no cycles in the service implementation dependency directed graph. Since this information refers only to direct dependency we call the function representing this information *dependency*1.

$$dependency1 : Service \times Service \rightarrow \Re^{+}$$

For example, *dependency*1(Travelocity, Avis) = 2.2 as each call to Travelocity results in the Travelocity service calling the Avis service an expected 2.2 times.

We use this information to estimate the complete dependency matrix, the number of calls generated, directly or indirectly, by a single call to a service on every other service. We estimate the complete dependency matrix by computing the transitive closure of the *dependency*1 matrix. We do this to allow for a concise definition of the service dependency information and because collecting multi-level information of this sort in a distributed system is quite difficult. However if the complete graph has been measured, it should certainly be used in preference to the estimate.

$$dependency : Service \times Service \rightarrow \Re^{+}$$

This is the same approach used in gprof in estimating call graph values [10].

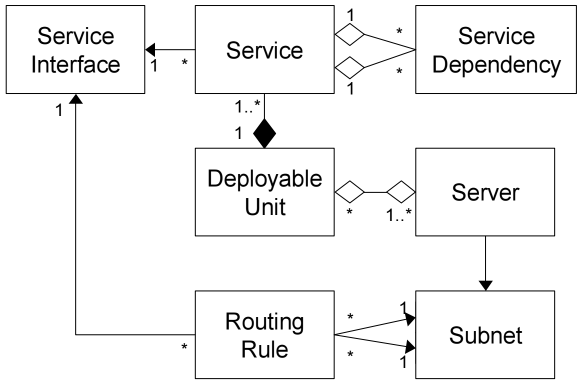


Fig 1: A UML Diagram of the Service Oriented Model

Client Service. We define a distinguished client service whose function is to invoke the externally accessible services. The client service makes the correct mix of requests that match the expected calls from all the system's clients. The client service is special in that we do not attempt to model its internal behavior or allocate resources to it. The distinguished client service gives us a single row of the dependency matrix to concentrate on that defines the expected call load that we are expecting for each service. Since there are no calls to the client service, one should think of the counts in the client row of the dependency matrix as representing the number of calls on the indicated services by external clients per unit time.

Formally, *client* is simply a distinguished element in *Service*.

$$client \in Service$$

Harking back to our car rental example, we can define the client service as making 400 calls per minute on the Travelocity service, 300 calls per minute on the LastMinute service, 100 calls per minute on Hertz and no calls on Avis. Assuming that the Hertz and Avis services make no service calls themselves, then taking the five services in the order: *client*, Travelocity, LastMinute, Hertz and Avis, we have a first level dependency matrix given as:

$$\begin{pmatrix} 1 & 400 & 300 & 100 & 0 \\ 0 & 1 & 0 & 1.4 & 1.4 \\ 0 & 0 & 1 & 1.7 & 2.2 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

The ones on the diagonal can be thought of as meaning that a call on a service results in one call on itself. We then take the transitive closure of this matrix to estimate the full dependency matrix. The top row of this dependency matrix gives us the frequency count of invocations of the corresponding service during the one-minute client interval:

$$\begin{pmatrix} 1 & 400 & 300 & 1170 & 1220 \end{pmatrix}.$$

These invocation counts can be combined with the preceding service load units to define the total execution load on the system. For example, if *loadU*(Travelocity) = 1000, *loadU*(LastMinute) = 1200, *loadU*(Hertz) = 6000, and *loadU*(Avis) = 7000, then the execution resource needed to meet the throughput requirements on the services are 400,000, 360,000, 7,020,000, and 8,540,000 units per minute respectively, and the total execution rate in the system is 16,320,000 units per minute.

In addition each service may have an availability requirement that defines the minimum probability that this service must be up and providing the needed service to the distinguished client service. We can define this requirement as a function that specifies the minimum probability that this service is allocated enough resources to perform its function. If there is no availability requirement on a particular service, the function may have value 0.

$$availableToClient : Service \rightarrow [0,1]$$

Note that this function is used along with the dependency information to generate the complete service availability function in the next section.

Deployable unit. Each separate service is not typically deployable on a server independently. A developer or administrator will typically build or configure a set of services into a deployable unit that can be installed on one or more machines. The developer may decide services need to be collocated in the same process or on the same machine to maintain efficiency or to reduce development time. When services are combined into a deployable unit, we do not model the dependencies between these services, instead the load and dependencies are rolled up into the services that is invoked externally. The form a deployable unit takes depends on the system being used. In J2EE a deployable unit might be represented as a preconfigured WAR, or web application archive, on Linux a deployable unit might take the form of a preconfigured RPM [11] file. Unlike an unconfigured WAR or RPM file which might contain a generic service implementation, a deployable unit contains all information to configure the implementation to take the role as a particular service, e.g., the data it will be accessing and the other services that it may need to contact.

Formally, the deployable units are just a set *Deployable* with a function *deploys* to represent the composition of a deployable unit out of its constituent services.

$$deploys : Service \rightarrow Deployable \,.$$

The composition of a deployable unit out of a set of services, modeled by this function, could have been generalized to a many to many relationship between services and deployable units, allowing a single service to be implemented by many different deployable units. In practice, practical engineering considerations cause administrators to avoid running more than one implementation of a single service, as this raises the probability of a failure caused by unexpected interactions between different deployable units running different implementations but implementing the same service. However, a service interface is typically implemented by many different services and thus potentially many deployable units.

Server. A server is an entity on which services can be executed. Servers are not referred to directly by applications; instead applications reference services that are automatically mapped to the servers on which they are deployed. Servers are typically hardware components, though servers can be constructed logically using virtual machine technology.

For each server we have a specified failure probability. This specifies the minimum long-term probability that the server is available and providing its full execution service. This is used in the next section to compute the probability that a service is available and providing service. We specify the server availability with a function:

$$serverAvailability : Server \rightarrow [0,1] \,.$$

Associated with each server is a rate at which it can perform load units, expressed in the same time units that were used for the client counts in *dependency*1 and the same load unit that was used for *loadU*. For example, a large multiprocessor server may be able to perform 50,000 load units per minute while a small server may only be able to support 1000 units per minute. We specify the execution rate of a server with the function *powerU*:

$$powerU : Server \rightarrow \Re^{+}$$

As stated earlier, a single power rating per server is a simplification. A more careful model may allow for service specific load ratings.

Resources on servers are assigned by the configuration system to deployable units. A unit may not consume more resources on a server than it is assigned. A single deployable unit may be deployed on many different servers simultaneously, in which case the load on the component services is divided among the servers, according to the ratio of resources assigned by the server to the deployable unit. We define the number of load units per unit time allocated to a deployable unit on a server as:

$$allocU : Deployable \times Server \rightarrow \Re^{+} \,.$$

Unlike the functions defined so far, this function is not defined by the administrator, but is instead an output of the optimization process. It specifies what services a server should run and the amount of server resources that should be assigned to each deployable unit. In the next section we develop constraints that will ensure that the allocation of resources to deployable units satisfies the system requirements. The resulting allocation must not overload the server, that is the following constraint must hold:

$$\begin{aligned}&\forall serv \in Server,\\ &\quad \sum_{\forall d \in Deployable} allocU(d, serv) \le powerU(serv).\end{aligned}$$

Subnet. A subnet represents a portion of the network containing a set of servers. Servers on the same subnet can communicate more cheaply, but servers on different subnets can be protected from each other by router based filtering and firewalls. We assume that each server is assigned to exactly one subnet. The assignment of servers to subnets will typically be an input to the configuration optimization process, though in other circumstances the assignment and creation of subnets might be an output of the process. We also distinguish the subnet from which client communications originate. This will typically represent the public Internet.

Formally, a subnet is just a set of items, *Subnet*, and a function subnet that assigns servers to subnets.

$$\begin{aligned}&subnet : Server \rightarrow Subnet\\ &clientSubnet \in Subnet\end{aligned}$$

Routing rule. The filtering that can take place between subnets is represented as a set of allowable service interfaces whose messages may pass between the subnets. Typically a routing rule will be assigned to a router or firewall to ensure that only the required communication can be passed and that this required communication is safe. Like the allocation of deployable units to servers, the configuration optimization process produces the set of subnet rules.

Formally the set of filter rules is a function, *rules*, from pairs of subnets to a subset of allowable service interfaces whose messages are allowed to pass from one subnet to the other.

$$rules : Subnet \times Subnet \to \wp(ServiceInterface)$$

3. PROPERTIES OF A SERVICE-ORIENTED SYSTEM

One measure of the usefulness of a model of a system is whether properties of the model can be defined that correspond to properties of the original system. In this section we present some interesting system properties that can be defined using our model and argue for their relevance. One interesting property that we do not define is latency. Reasoning about latency is a very tricky issue and we do not attempt to deal with it here.

Service Availability Requirement. A service's availability requirement is the probability that a service responds to a given request by one of its clients. A service's clients may include the distinguished *client* service as well as arbitrary other services that use this service. For a service to be available, in addition to the service itself being available all the service's dependencies must be available. Assuming that the availability of each request on each service is independent, we use the following constraint to define a *serviceAvailability* function that depends on the administrator-provided *availableToClient* as well as the *dependency*1 function.

$$\begin{aligned}&availableToClient(s1)\\&\quad\le serviceAvailability(s1)\\&\quad\quad\le \prod_{\substack{\forall s2\in Service\\ dependency1(s1,s2)>0}} serviceAvailability(s2)\end{aligned}$$

Informally this says that the service can be no more available than its constituents, but that it must be at least as available as any clients need it to be.

These constraints can be solved by starting with the services called only by the distinguished *client* service. Such a service is likely to have a nonzero value for the *availableToClient* function. This value can be factored to determine availability requirements for each of the services it calls. This process can be repeated until service availability requirements are derived for all of the services. As might be expected this process causes lower level services to have higher availability requirements.

Availability with Throughput. We define execution throughput and availability constraints simultaneously as for a service to be properly configured the probability that the service is meeting its throughput requirements must be as large as its availability requirement. Disconnecting availability and throughput admits of an implementation of a highly available system that while it may be nominally available, failures may have degraded its throughput so much that it may be considered dead by its users. One could complicate matters by defining multiple levels of availability, e.g. there is 99% probability that full throughput is available, but 99.99% probability that 50% throughput is available. But we do not do so in this paper.

An acceptable configuration must assign enough resources to each deployable unit so that with large enough probability all the services that are part of the deployable unit are getting enough execution resources to perform their function. We must also assign the resources in such a way that we never exceed the capacity of any server.

We can express the fact that a server may not be over allocated with the predicate:

$$\forall serv \in Server, \quad \sum_{\forall d\in Deployable} allocU(d, serv) \le powerU(serv)$$

This specifies that for all servers, that the sum of the load units allocated to each deployable is less than total load units provided by the server.

To form a predicate that insists that the needed throughput be provided with the required probability, consider a subset *S* of the *Server* set that represents the set of servers that are available at a moment in time. For each such subset $S \subseteq Server$ there is a well defined probability that exactly those servers are available. Assuming that the availability of each server is independent, that probability is given by:

$$\begin{aligned}setProbability(S) = &\prod_{\forall serv\in S} serverAvailability(serv)\\ \times &\prod_{\forall serv\in Server-S}(1 - serverAvailablility(serv))\end{aligned}$$

That is, the probability that exactly the set of servers *S* is available is the probability that each server in *S* is available times the probability that each server not in *S* is not available.

For each subset *S*, there is also an expression that represents the number of load units among the servers in *S* that are assigned to a given deployable unit, $d \in Deployable$. We compute this as a function *allocSU*:

$$allocSU(d,S) = \sum_{\forall serv\in S} allocU(d, serv).$$

For the set *S* to have adequate capacity to be classed as being available for *d*, the number of load units allocated to the unit *d* must be sufficient for performing the required load per unit time on the services making up *d*. We can compute this for a unit *d* by:

$$reqLoadU(d) = \sum_{\substack{\forall s\in Service\\ d=deploy(s)}} dependency(client, s) loadU(s)$$

that is, the sum, over all services that are part of the deployable unit, of the number of invocations on that service per unit time multiplied by the number of load units consumed by each invocation. This gives us the load units required per unit time, the same units as the allocation units for server resources assigned to a deployable in *allocSU*.

The availability of a deployable unit *d* in a given configuration is the sum over all subsets *S* of *Server* where the load units allocated to the deployable unit is sufficient to meet the execution requirements of the services that are part of the deployable unit, of the probability that the server configuration *S* exists. We define:

$$deployAvailable(d) = \sum_{\substack{\forall S \subseteq Server \\ allocSU(d,S) \geq reqLoadU(d)}} setProbability(S) .$$

If for each deployable unit this probability is larger than the maximum service availability requirement of all the services in the deployable unit, that is, when

$$\forall d \in Deployable, \quad deployAvailable(d) \geq \max_{\forall s: d = deploy(s)} serviceAvailability(s),$$

then the allocation of resources meets the availability and throughput requirements.

Security Distance. One of the important security considerations that must be taken into account when building a service infrastructure is router and firewall configuration. There are some services in which considerable skill and attention have been lavished in making sure that the service is ready to withstand the slings and arrows of outrageous hackers and other services which, while nominally secure, had best not be accessible to outside users. There are also services that store such sensitive data and best practices dictate that they should be locked away behind many levels of firewall.

One simple way of rating network service security is by the minimum number of subnet hops needed to get from the attacker to the target service. Each step along such a shortest path represents a subnet that has to be traversed and presumably hacked, in order to reach the target service. For example, in many web application infrastructures the service network is divided into 5 successively deeper subnets as illustrated in Fig. 2: a content subnet, a UI subnet, a business logic subnet, a database subnet and a SAN subnet. Each deeper level provides a lower level abstraction with less fine grain access checking and often less secure authentication. Accessing each successive subnet also requires hacking a different set of systems, typically using a different set of techniques. Note the NOC subnet, used for monitoring and administration, has connectivity to all the other subnets and if misconfigured can provide a shortcut access path into the deepest levels of the system, e.g. if servers in the NOC can be accessed from the Internet, sometimes allowed so that administrators can work from home.

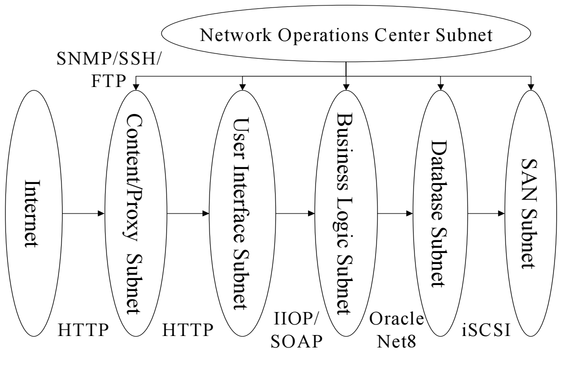


Fig. 2: Typical Subnet Structure

One way to increase the security level of a service to infinity is to make it inaccessible to clients by only running the service on servers that are on subnets with no direct or indirect path from the client. This makes these services secure, but unfortunately, also unavailable to the rest of the public.

When configuring routing rules it is important to allow communication between subnets where it is needed, e.g. services running on those subnets need direct communication, but at the same time we want to insist that certain services be run on servers that are deeply hidden from clients, that is there is a large security distance between the service and the attacker.

Network subnet distance is a simplification of the security restrictions one might contemplate, but it is a reasonable start and it mirrors current best practices [16].

The constraint on the existence of rules allowing all needed communication can be stated as:

$$\begin{aligned}
&\forall s1, s2 \in Service : dependency1(s1, s2) > 0 \Rightarrow \\
&\forall serv1, serv2 \in Server : \\
&\begin{pmatrix} allocU(deploys(s1), serv1) > 0 \\ \wedge\, allocU(deploys(s2), serv2) > 0 \end{pmatrix} \\
&\Rightarrow interface(s2) \in rules(subnet(serv1), subnet(serv2))
\end{aligned}$$

which states that for all pairs of services that communicate, and all servers that are assigned to run those services, then the interface those services use to communicate must be present in the rules set of the router that connects the two subnets.

Given the rule above, the security distance between two services, which we will denote as *securityDistance*(*s*1*, s*2), can be defined by the following recurrence.

First we define a predicate *connected* that determines whether there is a direct communications link between the two services, that is, whether any of the servers assigned to the services are on the same subnet.

$$\begin{aligned}
&connected(s1, s2) = \\
&\quad \exists serv1, serv2 \in Server : \\
&\qquad allocU(deploys(s1), serv1) > 0 \\
&\qquad \wedge\, allocU(deploys(s2), serv2) > 0 \\
&\qquad \wedge\, subnet(serv1) = subnet(serv2)
\end{aligned}$$

Then we can define the security distance with the following recurrence:

$$securityDistance(s1, s2) = \begin{cases} 0, & \text{if } connected(s1, s2) \\ 1, & \text{if } dependency(s1, s2) > 0 \wedge \neg connected(s1, s2) \\ \min_{\forall s3 \in Service} \{ securityDistance(s1, s3) \\ \qquad + securityDistance(s3, s2) \}, & \text{otherwise} \end{cases}$$

The security distance computed in this way can be used in constraints to insist that a sensitive service be a large distance from the client subnet. This can be used to restrict the optimizer from doing something silly like running a database service on the network DMZ in order to take advantage of its lightly loaded servers.

Data Risk. In another paper by this paper's authors [5], a security metric based on the aggregate risk of having data from different customers make use of the same device is defined. For example, a storage service provider may decide to store data from a single commercial bank on a storage unit and to accept a level of risk *r* in making that assignment while adding an airline's data to that storage unit may increase the risk of the assignment by a small amount but adding a competitive bank to the same unit may raise the risk considerably.

In the service-oriented context, a similar measure of data risk can be defined that quantifies the risk of placing deployable units on the same server or on the same subnet. The risk depends on the assurance level or trust we have in the server or the subnet's ability to keep the data separate and the risk associated with the information being accessed from the dependent services.

This metric was not used in the OPL implementation described in section 5, but was used in a separate OPL model described in [5].

Network Bandwidth. The network bandwidth used in a system can sometimes be an important consideration in system design. The internal switching inside a subnet is generally implemented by high performance switching equipment that has been optimized for network performance. Communication between subnets is performed by routers that have been optimized for security and for implementing many hundreds of complex filtering rules. Limiting the load on these expensive routers can sometimes be an important consideration.

To help express constraints or optimizer objective functions dealing with bandwidth, we define a new *traffic* function. The value *traffic*(*sn*1, *sn*2, *interface*) reports the number of invocations per unit time of the given interface that may travel between the given subnets. If the subnets are equal, the function gives the amount of intra-subnet traffic using the given interface. This function can be used to define constraints or minimize the usage of network traffic.

First we define the function *runsIn* that computes the set of subnets used for executing a given service:

$$runsIn(s) = \bigcup_{\substack{\forall serv \in Service: \\ allocU(deploys(s), serv) > 0}} subnet(serv)$$

Given this function we can define the traffic function as:

$$traffic(sn1, sn2, i) = \sum_{\substack{\forall s1, s2 \in Service: \\ i = implements(s2) \\ \wedge sn1 \in runsIn(s1) \\ \wedge sn2 \in runsIn(s2)}} dependency(client, s1) \times dependency1(s1, s2)$$

As can be seen this sums over all pairs of services where the second service implements the given interface and the services run on the given subnets. For each pair we look at the expected number of service invocations of the given type that will be requested per unit time. This is given by the expected number of invocations from the *client* to service $s1$ times the number of invocations that $s1$ makes directly to $s2$.

4. OPTIMIZING A SERVICE-ORIENTED SYSTEM

In the previous sections we have seen how to describe a service-oriented system and how to define properties and constraints on a service-oriented system; we can now look at optimizing a service-oriented system. In mathematical programming, optimization is driven by an objective function.

The difference between an objective function and a constraint is that a constraint must hold in order to have a solution, while the objective function is merely optimized from among the solutions meeting all the constraints. While there can be many constraints in a constraint satisfaction problem, there can only be one objective function.

Some useful objective functions include those for:

– Minimizing the cost of the system. The cost of a system is generally a linear formula involving the number and cost of each server and perhaps the number of subnets. Meeting requirements with the least cost is a common objective in optimization. In this case the objective function may be the number of servers that have not been allocated to any deployable unit. That is to maximize:

$$\left|\{serv \in Server : \forall d \in Deployable : allocU(serv, d) = 0\}\right|$$

– Maximizing the security of a service. In addition to setting minimum security distance constraints, the administrator may be looking to maximize the minimum security distance from an attacker subnet to a given set of services. That is, given a priority set of services, *Protected*, we might want to maximize

$$\min_{s \in Protected} securityDistance(clientSubnet, s).$$

– Maximizing the capacity of a system. If the load on the services may grow unexpectedly, the administrator may wish to build a system out of an existing hardware base that can respond quickly to unexpected spikes in demand by spreading any extra capacity evenly throughout the service deployments. We can compute the percentage of over capacity allocated to a service and attempt to maximize the minimum level of over capacity over all the deployable units by maximizing:

$$\min_{d \in Deployable} \frac{allocSU(d, Server)}{reqLoadU(d)}.$$

– Minimizing the number of routing rules. Routing rules consume resources on a router and having too many rules can cause the router to become overloaded, usually causing operators to ill advisedly remove rules. If an organization's routers are on the edge, minimizing this objective function could be important:

$$\max_{sn1, sn2 \in Subnet} \left| \bigcup_{i \in Interface} rules(sn1, sn2, i) \right|.$$

There are many other objective functions that can be defined. This list is just meant to be illustrative.

5. IMPLEMENTATION EXPERIENCE

The formulas given here can be entered nearly unchanged to build an OPL, Optimization Programming Language, application. In this section we describe a few changes that were made to facilitate expressing these formulas in OPL and how we reduced the search space to get faster results. We also describe a realistic example that we used to test the feasibility of the approach.

5.1 OPL Model Changes

A first implementation of the OPL model was made that used the functions as defined here directly. Unfortunately this model did not scale to larger configurations. The primary problems were the large solution space for the *allocU* function, the non-linearity in the probability function *setProbability* and the difficulty in entering the large amount of data for hundreds of nearly identical servers.

To speed up the search and ease the data entry we made two simplifications to the model given above:

– We assume that the allocation of load units to a deployable unit is identical on all of the servers it is assigned to. This reduces the search space considerably without removing too many interesting solutions. This is an acceptable simplification as a big disparity between allocations for the same deployable causes difficulties in reaching availability goals as the failure of the server with the largest allocation for a given deployable causes an undue loss for that deployable, making achieving availability much more difficult.

– We group the servers into server classes of identical servers; all the servers in a class have the same availability, power units and are on the same subnet. Each server class might correspond to a rack of servers in a data center.

– We assign deployable units only to servers in the same server class. This simplifies the computation of the *runsIn* function above, makes the security limits easier to reach, and reduces the solution search space considerably, again without removing many important configuration possibilities. Such a server assignment is typical of service assignments inside a cluster in a single data center. Services that are replicated across a WAN and logically run on different subnets must be modeled as separate services implementing the same service interface. In most cases this is appropriate as, given WAN communication costs in general and the costs of single copy serializablility in specific, the services are not really identical and are assigned to data centers manually.

By making these changes we can greatly simplify the computation of the products in the *setProbability* function by making use of the binomial theorem. We can do this because each *serverAvailability* probability used for a deployable is identical, so instead of summing over the power set of servers available, we sum over the number of servers available. Similarly the function *allocSU* is simplified because the *allocU* function values are either identical or zero for each server being used by a deployable unit.

For a particular deployable unit *d* that is assigned to *deployU*(*d*) units on *dcnt*(*d*) servers all in a server class with availability *avail*, the probability that the system is available with enough resources is:

$$\sum_{\left\lceil \frac{reqLoadU(d)}{deployU(d)} \right\rceil \le w \le dcnt(d)} \binom{dcnt(d)}{w} avail^{w} (1 - avail)^{dcnt(d)-w} .$$

5.2 OPL Implementation

An interesting feature of the OPL programming model is the six different sub-languages it supports, five of which are used in this application:

– A data declaration language that is used to define the form of the input data, the intermediate data, and the search space of the variable data.

– A sequential initialization language that is used to compute the values for the intermediate data based on the input data. Intermediate data so computed is considered to be 'ground' and can be used in more contexts than the variable data. In our application this is used to compute *dependency* from *dependency1* and to compute *serviceAvailability* based on *dependency1* and *availableToClient*. The former is straight forward, but the latter involves spreading probabilities.

The approach taken is to consider each target service in an order compatible with the *dependency*1 relation and assign that service a computed availability. We do this by looking at all the up-level services that the target service is used by and taking the maximum availability requirement implied by each such up-level service. If the up-level service has an availability requirement either specified or computed, we then look at all of the subsidiary services used by the up-level service. Some of those subsidiary services may precede the target service in the ordering and thus may already have a computed availability, these availabilities are divided out of the target availability. The remaining probability is spread evenly among those without specified availability by taking the appropriate root of the availability.

– A data instantiation language that is used to provide the input data and thus define a particular instance of the model to be solved. Data items instantiated in this language are stored in a separate file from the other items, facilitating using the same model for many different similar problems.

– A first order logic based constraint language used to specify the constraints relating the input data, the intermediate data and the output data.

– A backtracking based search control language that is used to control the search through the output space. This language is the most problematic as this has to be modified when a new optimization criterion is chosen. A subtle change in strategy can make the difference between search that won't end for many lifetimes and a result that is generated in minutes.

The commented OPL application is available on the authors' web site [13].

5.3 Realistic Test Case

To test the usefulness of this approach we wanted to apply the model to a realistic test case. The problem is NP-hard, so one can certainly find problem instances which cannot be solved in a reasonable amount of time. However we wanted to pick a test case that might come up in practice and see how this approach worked on this example.

In this test case we defined a configuration consisting of 26 services in 17 deployable units, with 8 different service interfaces, deployed on 160 servers in 8 different server classes running on 5 subnets. The availability of the servers varied from 3 nines of availability (i.e. 99.9%) to four nines. The service availability

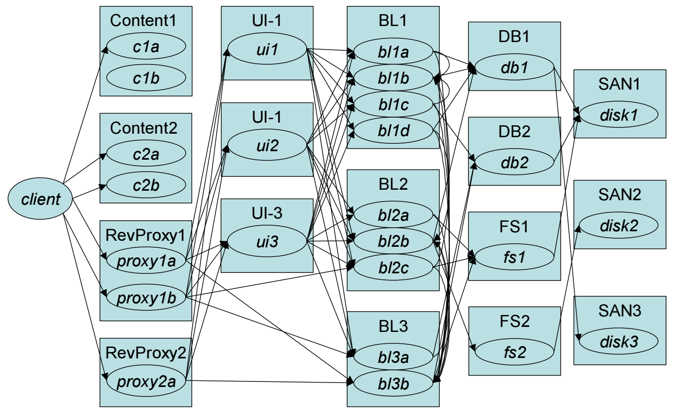


Fig. 3: Test Case Services, Deployables and Dependencies

requirements of the top-level services varied from three to four nines. The derived service availabilities for the deeper services went up to five nines and these deeper services were constrained as needing a security distance from the client of at least 3. We set the objective function to maximize the minimum level of over capacity from among the 17 deployable units.

The services were designed to model a modern multi-tier web based system consisting of client accessible static content and reverse web proxy services fronting for an inner tier of application services providing the application UI control and page generation. The UI services were then built on a tier of business logic services. Unlike the other service layers the business logic services are available both from the proxy layer and from the UI layer and they have a rich degree of interconnection that is difficult to see in the diagram. The services have been factored to remove any cycles from the dependency graph. The business logic services in turn build on a set of file and database services, which are in turn built on a set of virtual disk services implemented by a storage area network. The services, their grouping into deployable units and the dependencies are illustrated in Fig. 3.

Considerable tuning in the search procedure was needed to order the configurations tested so that the progress towards a solution progressed at a reasonable rate. At this point, OPL is able to find acceptable solutions after running for several minutes on a single 1.5 GHz processor. Finding optimal solutions for nontrivial objective functions is more elusive as the entire solution space often has to be searched, taking over 10 hours for the sample problem. For many uses this performance is adequate, for example, in configuring an enterprise data center for a new application or an application service provider for a new customer. For other uses, such as online reconfiguration after a device failure or configuring a dynamic grid computer, this performance is not adequate.

Note that a numerical instability in the availability computations currently limits the number of servers per server class to 21.

This result indicates that this approach to solving configuration problems is promising; though much more work remains to be done to show that it is practical and efficacious.

6. RELATED WORK

A modeling based approach to quality of service prediction is standard fare in queuing theory, but the focus is generally on the much more difficult measure of response time, a measure we leave out of our analysis because of its complexity. However the typical server graph used in queuing theory carries over to the dependency graph used here.

Other attempts have been made to model quality of service properties of distributed systems, most recently in the context of a service grid [1], but many fewer properties are being optimized for. Other current work on service grids is focused on mechanism of configuration rather than the optimization of configurations.[2]

The most closely related work to this has been done in the area of provisioning of storage in a storage network. Data storage and services are closely related, and in fact one can think of data access as a special case of service provisioning, where it happens that the services allow for data access. Work done in this area includes innovative work done at HP [2][3][9][15] in configuring storage systems. The authors have made their own forays into storage management in [5][12].

Other related work lies in network provisioning, where resources needed to provide the required quality of service are reserved in advance. In this work the model is more based on dynamic load rather than the static load model used in this work. Examples include [6][7].

A subset of the service provisioning problem being considered here was addressed using constraint satisfaction in [8], but the problem was simple enough that the big guns of constraint satisfaction was not necessary for the solution.

The SmartFrog [9] system from HP provides tools for describing and deploying configurations.

7. CONCLUSION AND FUTURE PLANS

The approach of producing an abstract model of a complex system and reasoning about that abstract model is an oldie but a goodie. In this paper we have applied this technique to the problem of configuring services on a network of servers. We have shown how to compute properties of the resulting network and to use those properties to drive the automatic optimization of that network to meet a set of requirements defined over those properties.

There have been advances in constraint systems and automated reasoning in recent years and using these techniques to design and maintain complex computing systems is an opportunity ready to be grasped.

A necessary future step for this research is to experiment with configuring real systems to verify that the promised gains are actually achievable. This can also be used to determine if there are constraints missing in our model that allow the production of flawed configurations.

Another area for extension is the development of new types of security measures. Here we explore a simple security distance metric, but many other types of threat can be defined, e.g. the information risk measure used in [5]. Our security distance metric can be refined by allowing each routing rule to have a separate breakage cost, instead of the unit cost used here. The attacker would search for the lowest cost path to the inner systems. In addition the rules can be arranged in a partial order to represent which rules are implicitly broken when another rule is hacked. This can be used to model the fact that once a successful attack on a system is found, the same attack can be used against similar systems with no additional cost.

In this paper we define availability as a service having enough available resources to perform its function. This definition does not mean that the service has those resources for a long enough contiguous interval of time to actually *perform* its function. For example, a diabolical highly available server with a very short MTBF but an incredibly small MTTR, may provide high availability using our definition, but unacceptable performance in real situations. We would like to define service availability as the probability that a given request is successfully processed however, this doesn't easily match up with the definition of availability for a server, which is necessarily time based. We are looking into this issue.

Incremental configuration, finding the minimal reconfiguration of an existing system to adapt to a new set of requirements or a new environmental condition, is also a focus for future work.

8. ACKNOWLEDGEMENTS

Thanks to Aad van Moorsel and the anonymous referees for their insightful comments.

9. REFERENCES

[1] Rashid Al-Ali, Abdelhakim Hafid, Omer Rana1 and David Walker, An Approach for QoS Adaptation in Service-Oriented Grids, Concurrency Computation: Practice and Experience, 2004, 16(5)

[2] G.A. Alvarez, E. Borowsky, S. Go, T. H. Romer, R. Becker-Szendy, R. Golding, A. Merchant, M. Spasojevic, AVeitch, and J. Wilkes. Minerva: an automated resource provisioning tool for large-scale storage systems. ACM Transactions on Computer Systems, November 2001.

[3] E. Anderson, M. Hobbs, K. Keeton, S. Spence, M. Uysal, and A. Veitch. "Hippodrome: Running Circles around Storage Administration." In Proceedings of the Conference on File and Storage Technologies, January 2002.

[4] Argonne National Laboratory. The globus project. See Web Site at: http://www.globus.org/

[5] B. Aziz, S. Foley, J. Herbert, and G. Swart. "Configuring Storage Area Networks for Mandatory Security." In Proceedings of the IFIP WG 11.3 Working Conference on Data and Application Security, July 25-28, 2004.

[6] R. Balter, L. Bellissard, F. Boyer, M. Riveill and J.Y. Vion-Dury, "Architecting and Configuring Distributed Applications with Olan", In Proc. IFIP Int. Conf. on Distributed Systems Platforms and Open Distributed Processing (Middleware'98), The Lake District, 1518 September 1998.

[7] Shigang Chen and Klara Nahrstedt. An Overview of Quality-of-Service Routing for the Next Generation High-Speed Networks: Problems and Solutions, IEEE Network, Special Issue on Transmission and Distribution of Digital Video, Nov./Dec. 1998.

[8] O. Martín-Díaz, A. Ruiz-Cortés, A. Durán, D. Benavides and M. Toro, "Automating the Procurement of Web Services" In Proceeding ICSOC 2003. LNCS. Springer Verlag, 2003.

[9] Patrick Goldsack, Julio Guijarro, Antonio Lain, Guillaume Mecheneau, Paul Murray, Peter Toft, "SmartFrog: Configuration and Automatic Ignition of Distributed Applications," 2003 HP Openview University Association conference, See http://www.hpl.hp.com/research/ smartfrog/papers.

[10] S.L.Graham, P. B. Kessler, M. K. McKusick, "gprof: A Call Graph Execution Profiler" In Proceedings of the SIGPLAN '82 Symposium on Compiler Construction; SIGPLAN Notices; Vol. 17, No. 6, pp. 120-126, June 1982.

[11] RPM Package Manager. http://www.rpm.org.

[12] G. Swart, "Storage Management by Constraint Satisfaction," In Proceedings of the Workshop on the Immediate Applications of Constraint Programming held as part of the Ninth International Conference on Principles and Practice of Constraint Programming (CP'03), 29 September 2003.

[13] G. Swart, "Backup material for Automatic Configuration of Services." http://www.cs.ucc.ie/~gs1/ServiceConfig.

[14] Pascal Van Henteryck. OPL; Optimization Programming Language. The MIT Press, 1999.

[15] Julie Ward, Michael O'Sullivan, Troy Shahoumian and John Wilkes, "Appia: automatic storage area network design." In Proceedings of the Conference on File and Storage Technologies, January 2002.

[16] Elizabeth D. Zwicky, Simon Cooper, D. Brent Chapman *Building Internet Firewalls*, O'Reilly & Associates; 2nd edition (January 15, 2000)

Portlet usability model

Óscar Díaz
ONEKIN research group
Dpt. Of Computer Science
University of the Basque Country
(Spain)
34 943 01 8064
oscar@si.ehu.es

Coral Calero, Mario Piattini
ALARCOS research group
Dpt. Of Computer Science,
University of Castilla-La Mancha
(Spain)
34 926 29 5300
{Coral.Calero, Mario.Piattini}@uclm.es

Arantza Irastorza
ONEKIN research group
Dpt. Of Computer Science
University of the Basque Country
(Spain)
34 943 01 8064
jipirgoa@si.ehu.es

ABSTRACT

Emerging portlet standards (e.g. WSRP) promise to achieve true portlet interoperability. This will certainly fuel portlet syndication, and facilitate a market for portlets in the long run. As with other component technologies, this requires the existence of quality models that assist in ascertaining the provider that better fits the consumer needs. Usability is one of the characteristics defined in the ISO 9126 standard. This paper introduces a usability model for portlets. The model identifies the distinct subcharacteristics for portlet usability, and proposes distinct attributes and metrics to assess these subcharacteristics.

Categories and Subject Descriptors

D.2.8 [**Metrics**]

D.2.9 [**Management**]: Software Quality Assurance

General Terms

Management, Measurement

Keywords

Portlet, usability model, metrics

1. INTRODUCTION

A portlet is a multi-step, user-facing application to be delivered through a web application. They are very much like Windows applications in a user desktop in the sense that a portlet renders markup fragments that are surrounded by a decoration containing controls. Portals as application-integration platforms have rapidly embranced this technology, and they are currently the most notable portlet consumers ([1])

So far however, the lack of a common model prevented portlet interoperability. However, the recent delivery of the *Web Services for Remote Portlets* (WSRP) specification ([20]) promises to overcome this problem. This will certainly promote portlet syndication, and facilitates a market for portlets in the long run.

As with other component technologies, a portlet market requires the existence of quality models that assist in ascertaining the portlet provider that better fits the consumer needs.

Portlets can be considered as the evolution of two main movements in computing: componentware and distribution. Different quality models have been proposed for components [2, 19, 16, 8, 3, 4, 18] and for distributed systems [7, 20, 5, 15] however, there is not specific work developed for portlets.

Building a quality model is a complex undertaking ([6], [14]). Normally, the software product quality is hierarchically decomposed into characteristics and sub-characteristics which can be used as a checklist. The ISO 9126 ([11]) quality model is a case in point. The ISO/IEC9126 quality standard ([11]) states some general quality characteristics, which are further refined into sub-characteristics, which are decomposed into attributes. The values of the attributes are computed using some metrics. As the metrics are obtained for a specific quality sub-characteristic, this model can then serve the user to assess the product that better fits his/her quality preferences

This hierarchical model must be tailored to specific domains and could be used in conjunction with the ISO/IEC 14598-4 for software product evaluation ([10]). This norm sets the ground for several models of quality for different products and software processes.

This paper focuses on usability, one of the ISO 9126 characteristics. In our case, a product-based view is adopted ([9]) based on the opinion that the clearer we are about what to achieve in terms of product quality, the easier will be to tune the process accordingly ([6]). This view is also backed by ISO ([11]), when stated that evaluating a product can provide feedback to improve a process.

Usability measures software attributes which are related to its operation, with regard to easiness of use and adaptation of new operators ([11]). This definition should be particularized for the portlet case. As portlets share features with web application and components, usuability measures for these two areas are revised and adapted for the portlet case. Specifically, the quality model for web applications presented in [16], and the quality model for components introduced in [19], are taking as the starting points. Sub-characteristics are then introduced, replaced or eliminated to better cater for the new usability considerations.

From a portlet viewpoint, two users can be considered, namely, the portlet consumer (e.g. the portal master), and the end-user that interacts with the portlet. This paper focuses on the portlet consumer.

The next section presents a brief introduction to portlets. Section 3 introduces the portlet usability decomposition into sub characteristics and attributes and the metrics proposed for each sub characteristic. Section 4 summarizes and concludes this paper.

2. A BRIEF ON PORTLETS

A portlet is a multi-step, user-facing application to be delivered through a Web application (e.g. a portal). Its novelties could be better assessed by comparing portlets with an already well-known technology as Java Servlets. Similar to Servlets, Java portlets run in a portlet container, a server container that provides portlets with a running environment. Servlet generates HTML pages as a result of a browser invocation. Likewise, a portlet also generates XHTML fragments (or other markup language) to be framed by the invoker.

There are however, two main differences. First, a Servlet is a one-step process while a portlet comprises a multi-step process. Unlike Servlets, portlets support complete, full-fledged, web applications. Second, a Servlet can generate a full page. By contrast, a portlet generates fragments to be assembled with other portlets' fragments to build up the final page. This fragment is then included within a portal page, with very few changes to be made by the portal. Hence, a portal page can contain a number of portlets that users can arrange into columns and rows, and minimize, maximize, or customise to suit their individual needs. They are very much like Windows applications in a user desktop, in the sense that a portlet renders markup fragments that are surrounded by a decoration containing controls.

So far however, the lack of a common model prevents portlet interoperability. This impedes a portlet developed in, lets say, Oracle Portal, being deployed at a Plumtree portal, and vice versa. However, the recent delivery (2003) of the *Web Services for Remote Portlets* (WSRP) specification ([20]) promises to overcome this problem. WSRP uses WSDL for portlet specification. See [22] for an introduction to Portlets.

3. THE PORTLET USABILITY CHARACTERISTIC

From a portlet viewpoint, usability refers to the capability of the portlet to be understood, learned or used (i.e. invoked) when "used" under specified conditions. Here, it is important to notice that two kinds of users can be considered: the portal administrator and the end-user. This work focuses on the portal administrator which should be referred as "the user" hereafter.

As indicated in [11], usability metrics should be able to measure software attributes related to its operation, with regard to easiness of use and adaptation of new operators. This definition influences on the sub-characteristics we have identified for the portlet usability.

Next subsections present the usability sub-characteristics considered together with the metrics proposed for each one. For the definition of each characteristic, we have used indifferently the terms presented in [11], [16] or [19].

3.1 Understandability

Understandability can be defined as the capability of the portlet to enable the user to understand **what the portlet is about**. So, it bears on the users' effort for recognizing the logical concept and its applicability.

Following [11], *understandability metrics should be capable of evaluating the behavior of users without previous knowledge on software operation and measuring their difficulty on understanding software functions, operation and concepts. On doing this, it may be considered entities such as documentation (in all available formats, as on-line or printed), software interface and vocabulary*.

Component metrics introduce the programmatic interface as part of the documentation of the component. However, WSRP portlets have a generic interface to achieve interoperability (similar to Servlets). So, the interface as such does not give us any hint about the aim of the portlet. The WSRP API includes two functions *getPortletDescription()* and *getPropertyDescription()*, that give some details about some configuration parameters (e.g. locales, mime types, etc).

However this is not enough. Assessing the functionality of a portlet requires more than the configuration options available. In the component realm, additional information is provided through documents. Here, this option is also available.

Moreover, portlets offer an additional mechanism, i.e. the modes. A mode is way of behavior. Both the content and the activities offered by a portlet depend on its current mode. For instance, in the *"view"* mode, the portlet renders fragments which support its functional purpose (e.g. booking a flight seat). This is what we normally mean by interacting with a traditional Web Application.

Additionally, WSRP contemplates other modes. The one of interest in this context is the *wsrp:help* mode. When in this mode, the portlet may provide help screens that explains the portlet and its expected usage. Some portlets will provide context-sensitive help based on the markup the end-user was viewing when entering this mode. When in this mode, all interactions are aimed at describing **what the portlet is about**. It does not achieve any functional goal.

In table 1 the attributes and metrics proposed for understandability are shown.

Table 1. Attributes and metrics proposed for understandability

Attribute	Definition	Metric	Metric Domain
Interface language	The portlet interface supports different languages	Number of languages supported by the Interface	Natural number
Help mode	The portlet may provide help screens that explain the portlet and its expected usage	portlet help mode support	Boolean
View mode	The portlet renders fragments	Portlet view mode support	Boolean

	which support its functional purpose		
Ad-hoc mode	The portlet supports other modes	Ad-hoc mode support	Boolean
Documentation/User manuals	The portlet vendor provides the portlet with documentation/user manuals in paper	Documentation degree	List (0..4): nonexistent, scarce, normal, complete, very complete
Vendors support	The portlet vendor gives support to the user	Vendor support availability	Boolean

3.2 Learnability

Learnability is the capability of the portlet to enable the user to learn **how the portlet achieves its aim**. Following [11], *learnability metrics should be capable of evaluating or drawing the user curve of performance on software operation, from a start point of no knowledge about the evaluated software. An external learnability metric should be able to measure such attribute as the behavior of user who is learning how to use the software.*

Traditional predictability attributes [8] can be used here e.g. on-line help, documentation, manuals and the like. However, the point to be underline here, is the existence of the *wsrp:preview* mode. In *wsrp:preview* mode, a portlet provides a rendering of its standard *wsrp:view* mode content, as a visual sample of how this portlet will appear on the end-user's page with the current configuration. This serves for the user to get familiarized with **how the portlet achieves its aim**.

3.3 Customizability

Customizability refers to the attributes of software that enable the software to be customized by the user, to reduce the effort required to use it and also to increase satisfaction with the software.

Table 2. Attributes and metrics proposed for learnability

Attribute	Definition	Metric	Metric Domain
On-line help	The portlet includes on-line help for guiding the user during the portlet utilization	On-line help	Boolean
Documentation/User manuals	The portlet vendor provides the portlet with documentation/user manuals in paper	Documentation degree	List (0..4): nonexistent, scarce, normal, complete, very complete
Predictability	Portlet interface icons are easily related to the actions the portlet performs	Portlet predictability	List (0..4): too difficult, difficult, normal, easy, too easy
Preview mode	The portlet may provide a	Portlet preview mode	Boolean

A first concern to be considered is the context or environment in which the software needs to be customized. Being a web application, portlets inherit the context properties contemplated for ubiquitous web application ([13]), namely,

- *Location:* This attribute copes with the need for mobile computing and location-aware services to capture information about the location from which an application is accessed. This is also related to the notion of Localization. *Localization* is the capacity to tailor one Web site to the idiosyncrasies of a given culture and it is becoming an increasing concern. The aspects of cultural diversity that need specific support are not only limited to the language, but to a range of topics from date and calendar issues to letter written figures or telephone numbers. These aspects known as *locales*, are normally arranged along two features, namely, *Language* and *country*. Different standardization efforts are being conducted to set the possible values, and thus we have ISO3166 and ISO639-2 standards. As part of its requirement list, the portlet should indicate the languages and countries it is expected to support.
- *Time.* This attribute addresses how the application adapts to certain timing constraints such as opening hours of shops or timetables of public transportation.
- *Device:* This attribute discusses the demand of ubiquitous web applications for any media, in terms of multi-channel delivery, and it provides basic information about the hardware and software capabilities of the device accessing the application. This feature can in turn be split into *Markup_Type* (e.g. *HTML*, *WML20*, or *VoiceXML*), and *User_Agent* (e.g. *Netscape702*, *msie60*, or *nokia7650*).
- *Network:* This attribute considers adaptation from the network viewpoint, and whether network context information e.g., bandwidth or package losses, affects the application.
- *User:* This attribute regards the need for personalization, i.e. how the user profile (e.g., demographic data, knowledge, skills and capabilities, interests and preferences, goals and plans) are considered by the application.

Additionally, a portlet should consider an additional context: the application in which the portlet is going to be framed. According to this context, new characteristics can be included:

- *Window states*. Web applications are designed for being rendered in a full page. A page is the unit of delivery. By contrast, portlets tend to be rendered together with other portlets. The portlet markup (known as *fragments*) is delivered together with other portlets in the same portal page. This implies that the space available for portlet rendering can be constrained by the consumer. The so-called *Window state* characteristic gives a hint about the space available for portlet rendering. Options contemplated by WSRP include: *normal,* indicates the portlet is likely sharing the aggregated page with other portlets*; minimized,* instructs the portlet not to render visible markup, but lets it free to include non-visible data such as JavaScript or hidden forms*;* *maximized,* specifies that the portlet is likely the only one being rendered in the aggregated page, or that the portlet has more space compared to other portlets in the aggregated page*;* and *solo,* denotes the portlet is the only portlet being rendered in the aggregated page. This attribute measure how many window states are considered by the portlet.

- *CSS*. The portal environment can include look-n-feel guidelines. All portlets' rendering should share some aesthetic guidelines to preserve the identity of the portal. If the fragments of the portlet cannot be altered then, no customization to the look-n-feel prescription of the consumer is possible. Alternatively, the fragments can be "parameterized" using Cascade StyleSheets (CSSs) ([16]) which are then instantiated by the consumer to its own values. The CSS attribute is a Boolean that measures the support of parameterized, WSRP-compliant fragments.

Customizability also includes the easiness with which customization parameters can be obtained. Adaptive systems obtain these parameters automatically (e.g. the user agent can be readily obtained from the *http* request). By contrast, adaptable systems require the intervention of the user. For the portlet case, these users can be the portlet consumer (e.g. the portal administrator) or the end-user. Provided the portlet can be customized to the given features, the portlet consumer has an API to customize the portlet to the consumer idiosyncrasies. As for the end-user, an *edit* mode can be available in the portlet for the end-user to introduce her preferences. The existence of this WSRP-compliant mode can then be seen as an attribute of the customizability sub-characteristic.

In table 3 the attributes and metrics proposed for customizability are shown.

Table 3. Attributes and metrics proposed for customizability

Attribute	Definition	Metric	Metric Domain
Location	The portlet captures information about the location from which it is accessed	Location availability	Boolean
Localization	The portlet indicates the languages and countries it is expected to support.	Language	Standard language list
		Country	Standard country list
Time	The portlet allows to adapt the application with respect to certain timing constraints	Time adaptation availability	Boolean
Device	The portlet can adapt itself to different hardware and software capabilities	Mark-up type	List
		User agent	List
Network	The portlet can adapt itself to different networks	Network adaptation availability	Boolean
User	The portlet takes into account the personal characteristics of the user	User_profile	WSRP structure
Window states	Space left for portlet rendering	Window state	(normal, minimized, maximized, solo)
CSS	The portlet consider aesthetic guidelines to preserve the identity of the portal	CSS availability	Boolean
Edit mode	A mode for the end-user to configure the portlet	Edit mode availability	Boolean

3.4 Compliance

Compliance is the capability of the portlet to adhere to standards, conventions or regulations in laws and similar prescriptions relating to usability.

Following [11], *an usability compliance metric should be able to measure an attribute such as the number of functions with, or occurrences of compliance problems, which is of the software product to failing to adhere to standards, conventions, style guides or regulations relating to usability which are required to be adhered.*

Currently, there are two main standards for portlets, namely, WSRP ([20]) that describes the interfaces and protocols that regulate the interaction between the portlet consumer and the portlet provider, and JSR168 ([12]) that faces portlet implementation in the Java world.

It should be noted that WSRP schemas can be extended to cater for special needs. Notice however, that these are ad-hoc extensions than can jeopardize portlet interoperability.

In table 4 the attributes and metrics proposed for compliance are shown.

Table 4. Attributes and metrics proposed for compliance

Attribute	Definition	Metric	Metric Domain
Interface Standards	The portlet follows the WSRP standard	WSRP compliance	Boolean
Extensions	The portlet includes some extension	WSRP extensions	Boolean
Implementation standards	The portlet implements a standard (as JRS 168 for Java)	JSR168 compliance	Boolean

4. CONCLUSIONS AND FUTURE WORK

The recent delivery of *WSRP* will certainly facilitate a market for portlets in the medium run that will *"make the Internet a marketplace of visual web services* (i.e. portlets)*, ready to be integrated into portals"* as stated in the *WSRP* proposal. This endeavor will be forwarded if quality models are available to ease the selection among portlet providers.

This work focuses on usability as a key ingredient of the quality model. The study is restricted to those aspects we have identified as distinctive of the portlet case. Other usability subcharacteristics such as *attractiveness* (i.e. *the extent of which user likes the software during operational testing, usability testing or user operation* ([11]) are not specific for portlets, and models proposed for web applications can be used here..

Once the definition of metrics will be complete for this and for the other quality characteristics, it would be necessary to validate them. The validation must be done in two ways. The first one is the formal validation of metrics using specific formal frameworks as the ones proposed by [17] or [21]. The second one is the empirical validation that could be done through surveys, controlled experiments and case studies. As a result of the validation process we will have a set of metrics useful for a given quality characteristic. Using these metrics it will be possible to construct a composition function for the calculation of each quality characteristic for a given portlet. The result of applying these functions to several portlets (which have the same utility) can be used to determine and to choose the best one for the user from the quality point of view.

After that and in addition to the quality characteristics of the product itself, it will be necessary to consider other factors like those relative to the service supplier (ISO 14598-4-5). For instance, the software engineering processes used, and the reputation of the portlet provider (e.g. financial stability, experience, capacities, availability, service maintenance, development plans, etc.). It rests to be seen what adaptations (if any) are needed to accommodate the models proposed for COTS to the portlet case.

5. ACKNOWLEDGMENTS

This work is supported by the Spanish Science and Technology Ministry (MCYT) and the European Social Funds (FEDER) under contracts TIC2002-01442 and TIC 2003-07804-C05-03). It also benefits from funding of "Consejería de Ciencia y Tecnología" of Junta de Castilla la Mancha (PCB-02-001).

6. REFERENCES

[1] Anuff, E. (2004) WSRP and the Enterprise Portal, http://www.syscom.com/story/print.cfm?storyid=44674

[2] Bertoa, M.F. and Vallecillo, A. "Quality Attributes for COTS Components". In Proc. of the 6th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE 2002). Málaga, Spain, June 2002.

[3] Botella P., Burgués X., Carvallo J.P., Franch X., Pastor J.A. and Quer C. (2003) Towards a Quality Model for the Selection of ERP Systems in Component-Based Software Quality. Cechich et al (eds). LNCS 2693. pp. 225-245

[4] Cai, X., Lyu, M.R., Wong, K-F. and Ko, R. (2000). Component-Based Software Engineering: Technologies, Development Frameworks, and Quality Assurance Schemes. Proc. of the Seventh Asia-Pacific Software Engineering Conference (APSEC'00), IEEE Computer Society, 372-379.

[5] Calero, C., Ruiz, J. and Piattini, M. (2004). A metric web survey using WQM. Proc. of the International Conference on Web Engineering (ICWE'04).

[6] Dromey, R.G. (1996). Cornering the Chimera. IEEE Software 20 (1), 33-43.

[7] Fernández, G. and Rossi, P: (2000). Measuring Distributed Software Quality: A First Step. Proc. of the Argentine Symposium on Software Engineering, 19-28.

[8] Franch, X. and Carvallo, J.P. (2003). Using Quality Models in Software Package Selection. IEEE Software 20 (1), 34 - 41.

[9] Garvin, D. (1984). What does "product quality" really means?. Sloan Management Review, 24.

[10] ISO (1999). ISO/IEC 14598-4. Information Technology - Software product evaluation - Part 4 Process for purchasers.

[11] ISO (2001). ISO/IEC 9126. Software Engineering-Product Quality. Partes 1 a 4. International Organization for Standardization/ International Electrotechnical Commission.

[12] JSR 168, Java Community Process. JSR 168 portlet specification, October 2003. http://www.jcp.org/en/jsr/detail?id=168

[13] Kappel, G., Pröll, B., Retschitzegger. W., Schwinger, W.(2003), Customisation for Ubiquitous Web Applications: A Comparison of Approaches. International Journal of Web Engineering and Technology, 1(1): 79-111

[14] Kitchenham, B. and Pfleeger, S.L. (1996). Software Quality: The Elusive Target. IEEE Software 20 (1), 12-21.

[15] Landor, P. (2003). Understanding the Foundation of Mobile Content Quality. A Presentation of a New Research Field. Proc. of the 36th Hawaii International Conference on System Sciences. IEEE Computer Society, 1- 10.

[16] Niessink, F. (2002) Software Requirements: Functional & Non-functional Software Requirements. www.cs.uu.nl/docs/vakken/swa/ Slides/SA-2-Requirements.pdf

[17] Poels, G. and Dedene, G. (2000). Distance-based software measurement: necessary and suffcient properties for software measures. *Information and Software Technology* 42, 1, 35–46.

[18] Sedigh-Ali, S., Ghafoor, A. and Paul, R. A. (2002). Metrics-Guided Quality Management for Component-Based Software Systems. Proc. of the 25th Annual International Computer Software and Applications Conference (COMPSAC'01), IEEE Computer Society.

[19] Simão, R.P. y Belchior A. (2003). Software Component Quality. In: Quality in Component Based Systems. Cechich, A. et al. (eds.). Springer-Verlag.

[20] WSRP . Web Service for Remote Portals (WSRP) Version 1.0, 2003. http://www.oasis-open.org/commitees/wsrp/

[21] Zuse, H. 1998. *A framework of Software Measurement*. Walter de Gruyter, Berlin.

[22] Díaz, O, and Rodríguez, J.J. (2004) Portlets as Web Components: An Introduction, *Journal of Universal Computing Science*, 10(4)

WS-QoC: Measuring Quality of Service Compliance

Ali Shaikh Ali
Cardiff University
Newport Road CF24 3AA
Wales, U.K
Ali.Shaikhali@cs.cf.ac.uk

Omer F. Rana
Cardiff University
Newport Road CF24 3AA
Wales, U.K
O.F.Rana@cs.cf.ac.uk

David.W.Walker
Cardiff University
Newport Road CF24 3AA
Wales, U.K
David.W.Walker@cs.cf.ac.uk

ABSTRACT

Web services have been the focus of much research activities in recent years, especially those that provide a virtual framework for resource sharing across institutional boundaries. As a consequence of this, we envision a service rich environment in the future, where service consumers are faced with the inevitability of selecting the "right" service. In such a scenario the Quality of Service (QoS) serves as a benchmark to differentiate between services. However, the autonomy of service providers implies that the provider may defect in the course of service delivery, and not accurately deliver the quality agreed upon within a Service Level Agreement (SLA). It becomes necessary, therefore, to measure how "trustworthy" a provider has been in complying with the agreed levels in the SLA in the past. We propose Quality of Compliance (QoC), which provides a mechanism for assessing the level of compliance of the service provider to an SLA, and therefore gives an indication of the actual service quality delivered. We also present our prototype implementation of WS-QoC.

Categories and Subject Descriptors

C.4 [**Computer Systems Organization**]: Performance of Systems—*Reliability, availability, and serviceability*

General Terms

Algorithm, Performance, Reliability, Measurement

Keywords

Trust, Reputation, Quality of Service

1. INTRODUCTION

The pervasiveness of Web services provides a novel form of communication between individuals and organisations, leading to new flexible work patterns and making organisational boundaries more permeable. Upcoming standards for the description and advertisement of, as well as the interaction with, Web services promises a seamless integration of business processes and applications over the Internet. As a consequence of the rapid growth of Web services, consumers are faced with the inevitability of selecting the "right" service. In such a scenario the Quality of Service (QoS) serves as a benchmark to differentiate services [20] – and thereby aid the selection process. In this paper we assume that the current rapid takeup of Web services is likely to lead to a service 'rich' environment, necessitating users to select between services offering related functionality.

In the context of Web services, QoS metrics determine the service usability and utility, both of which influence the popularity of the service. It comprises of techniques that aim to bring a balance between the needs of the service consumer and those of the service provider – while being constrained by the limited network and server resources [11]. Delivering QoS is a critical and significant challenge because of its dynamic and unpredictable nature. To ensure this QoS, the service consumer jointly with the service provider should define a Service Level Agreement (SLA) as part of a service contract. An SLA provides some guarantees about the likely behaviour of a Web service for the consumer. An SLA also defines the mutual understandings and expectations of service delivery between the service provider and service consumer. An SLA is often defined between a single provider and consumer – although there may be cases where both providers and consumers can be grouped – and the SLA is defined for the group.

Although, the main goal of the SLA is to enforce the service delivery based on various QoS properties, the parties in the SLA cannot always be assumed to be trustworthy to fulfil their obligations. Hence, a service provider which offers excellent QoS metrics for delivering a particular service might not accurately deliver the agreed levels of these metrics. There is no mechanism to quantify how trustworthy the service provider has been in previous service deliveries. The degree of trustworthiness has been loosely defined as the "reputation" of the provider in [1], [5], [19] and [9]. These authors quantify the reputation of the entities in their system based on plausibility consideration. However, plausibility consideration, which is contingent upon prior beliefs, fails to quantify the degree of trustworthiness in an entity. Obreiter [12] points out the limitation of plausibility consideration as follows: (1) the system assumes that the recommendation from a trusted entity is always correct (trustworthy); (2) the recommendations from newcomers are considered untrustworthy sometimes as there is no first hand experience with the recommendee and the recommendation behaviour of the recommender is unknown; (3) the plausibility consideration may be infeasible due to the lack of background information – although their recommendation might be correct; (4) recommendations can only be credibly passed on by commonly trusted entities. If the system lacks such entities, the recommender is solely in charge of the distribution of the issued recommendation; (5) the system lacks a formal method for the defamed entity to appeal and defend itself – which may lead to (6) doubts about the effectiveness of the reputation system. In this context, effectiveness refers to the result of pruning untruthful recommendations, and to the outcome of disseminating recommendations respectively. If there are doubts about the effectiveness of such pruning, the entities lack incentive for good behaviour.

Subsequently, neither the traditional QoS metrics nor the plausibility consideration serves as a benchmark to select the "right" service. In light of this issue, we introduce a new concept called the "Quality of Compliance" (QoC) to make sure that the "right" service is more appropriate to the intended (and expected) behaviour. The QoC of a provider (for a particular service) is defined as the degree of compliance for delivering the qualities of the service as defined in the SLA. Therefore, service selection should include the QoC metric along with the more widely used QoS metrics.

The remaining of this paper is structured as follows. Section 2 provides a survey of existing approaches for quantifying the trustworthiness of entities in distributes systems, and the existing findings for selecting services based on QoS metrics. A discussion of the relationship between QoC, QoS and Trust is provided in Section 3. Section 4 introduces a framework for integrating QoC in SLA-enabled Web services model.

2. RELATED WORK

2.1 Trust and Reputation

The significance of quantifying the trustworthiness of parties in distributed systems is evident from on-going research – especially in the context of file sharing systems such as KaZaA. There are many approaches introduced in the literature for assisting the user in identifying the trustworthiness of other entities. We categorise these approaches as follows:

(1) Collaborative Trust: The basic idea of collaborative trust, or "network of friends", is built on the assumption that if A does not have a trust relationship with B, then A asks his "friends" about their trust opinions or recommendations about B. Then A usually weighs the opinions by the trust that A has on the particular friend that returned a recommendation. A system that uses collaborative trust to retrieve the trust value of an entity is usually called a recommendation-based reputation system. An example of a collaborative trust system aforementioned is EigenTrust [9], a trust management system for Peer-to-Peer systems. In EigenTrust each peer i is given a unique global trust value that reflects the trust that the whole system puts in i based on past download experience. The system computes the global trust value of a peer i by computing the left principle eigenvector of a matrix of the normalised local trust values of all the peers in the network that had transactions with i. The local trust values of any peer towards i is basically the average number of satisfactory transactions it has conducted with i. By having peers use these global values to choose the peers from whom they download, the mechanism is effectively used to identify malicious peers and isolate them from the network.

(2) Rule-Based mechanisms: The general concept of this approach is to derive rules to determine whose recommendation is trusted. Rahman and Hailes [1] propose a model for the deployment of rules in virtual communities to allow agents to decide which others agents' opinion they trust more. In this way, agents can progressively tune their trust in other entities based on the outcomes of previous interactions.

(3) Transfer of Importance: The basic idea of this approach is that if A has a link to B, then a portion of A's importance is passed to B's. A well-known reputation system that follows this approach is PageRank [13], the core ranking system for Web pages performed by the `Google.com` search engine. The basic idea of PageRank (PR) is that the value associated with each page depends on the PR of the pages pointing to it. A page has a high PR value if there are many pages pointing to it, or if there are some pages pointing to it that have high PR values. The PR algorithm represents the structure of the Web as a matrix, and PR value as a vector. The PR vector is derived by computing the matrix-vector multiplication repeatedly.

(4) Measuring 'expertise' similarity: This approach is based on measuring the similarity in the competence of community members. This requires the maintenance of some type of user profile. Several multi-agent systems have been developed to serve this purpose. The agents in these systems act on behalf of community members, maintaining profiles or routing questions to other member agents. These agents use profile similarity as a criterion for finding possible partners. This approach is closely related to the MatchMaking mechanism – whereby agents with similar expertise can be grouped together to offer a service. See [18] for a discussion on expertise similarity.

Although these approaches are created to assist the user to identify the trustworthiness of other entities, they do not measure the real past performance of the entities as they are based on plausibility consideration. These approaches are mainly aimed at providing a subjective assessment of other service providers. Extending them with real measurements from a monitoring entity therefore is likely to lead to more reliable ratings.

2.2 Quality of Service

Selecting services based on Quality of Service has been the subject of intense research in recent years and has reached a certain degree of maturity. Rashid et al. [2] propose a system for discovering and selecting services based on QoS metrics. In their system, they use the `propertyBag` feature of the UDDIe registry [16] to publish QoS metrics in the registry. They also use the range-based search feature of UDDIe to search for QoS metrics based on numerical ranges, utilising operators such as "less-than", "greater-than" and "between", in addition to logical operators such as AND/OR. Tian et al. [17] propose another framework, WS-QoS to enable an efficient Web service selection based on application level QoS parameters. A broker service evaluates the QoS requirements of a client against the QoS capabilities of a service provider to find the most suitable service provider for each client. In this case, each service interface is annotated with particular QoS attributes that can be offered for that service. Some of these are specific to the service, while others apply for all services managed by a particular provider [3]. Web services such as:

Availability: The quality aspect of whether the Web service is present or ready for immediate use. It represents the percentage of time the server is available during an observation period. Larger values represent that the service is always ready to use, while smaller values indicate unpredictability of whether the service will be available at a particular time [11].

Accessibility: Accessibility is the quality aspect of a service that represents the degree to which it is capable of responding to a service request. It may be expressed as a probability measure denoting the success rate or chance of a successful service instantiation at a point in time. There could be situations when a Web service is available but not accessible. [11]

Accuracy: Accuracy defines the error rate produced by the service [14].

Payment Rate: Rate at which the service/transactions are charged [15].

Throughput: This metric represents the number of user requests that are handled by the system [14]. The response time of a Web service is related to its throughput. It is well known that the response time of a given system decreases as the system throughput increases.

Integrity: Integrity is the quality aspect associated with how the Web service maintains the correctness of the interaction in respect to the source. Proper execution of Web service transactions will provide the correctness of interaction [11].

Response Time: This is the most important QoS metric from a user's perspective. Response time measures the time interval between sending a request to execute a service, and the time that the response has been received by the user [15].

Latency: The time taken between the service request arriving, and the request being serviced [14]. The throughput of a system is related to its latency.

Performance: Performance measures quality aspect associated with a Web service – and is measured in terms of throughput and latency. Higher throughput and lower latency values represent good performance of a Web service [11].

Reliability: Reliability also measures the quality aspect of a Web service, and represents the degree of being capable of maintaining the service and service quality. The number of failures per month or year represents a measure of reliability of a Web service. Reliability is also defined as the probability that a request is correctly responded within the maximum expected time frame [11].

Regulatory compliance: Regulatory compliance measures conformance with some pre-defined (and agreed on) rules, law, standards, or established SLA [11],[14].

Security: Security is the quality aspect of the Web service of providing confidentiality and non-repudiation by authenticating the parties involved, encrypting messages, and providing access control. [11]

Some of these metrics can be directly measured – using monitoring tools that can be provided within the Web services hosting environment, while other metrics need to be derived from measured values. For instance, "Security" is a difficult metric to quantify – whereas "Latency" and "Response Time" can be directly measured.

Monitoring the compliance of service providers for the agreed QoS has also been introduced in literature. Bhoj et al [6] for example, describe an architecture that uses contracts based service level agreements (SLAs) to share selective management information across administrative boundaries. Although, the prototype implementation of this architecture has been used for automatically measuring, monitoring, and verifying SLAs for Internet services it does not provide a means for discovering services based on service providers' compliance metrics. In addition, the compliance report generated from the monitoring tool is meant to targets system administrators to analysis how well the system is conforming to SLAs. Thus, service requestors does not benefit from this information.

3. THE RELATIONSHIP BETWEEN QOC, QOS AND TRUST

QoS metrics have often served as the benchmark for selecting services. Identifying a service provider likely to deliver a specified QoS relies primarily in assessing the quality metrics supplied by the providers themselves. The current assumption is that service requesters trust these metrics. The autonomy of the service providers implies that the provider may defect in the course of the transaction, and not deliver the agreed levels of QoS defined and agreed upon before. Therefore, the QoS metrics supplied by the service provider are no longer trusted without a mechanism to prove that these metrics are actually being met. This verification can be undertaken during the course of a transaction, or once a transaction has completed. The QoC proposed in this paper provides a mechanism for assessing the level of compliance of the service provider – thereby giving an indication of the actual QoS delivered by a service provider.

The concept of QoC stems from the fact that there has been no practice of recording the achieved service levels once a transaction has been completed. Doing so gives an insight into the providers past performance by providing necessary data to progressively assess the compliance levels over a range of past transactions. QoC refers to the service providers' ability to meet the service level of each QoS metric specified in the SLA without incurring penalties. For example, the QoS metric 'response time' has a corresponding QoC value which gives an indication of the compliance of the service provider for the response time metric. Consider the following scenario, if a user agrees with a service provider to invoke an operation for a 100 times with the following QoS metric: response time = 10 seconds for each invocation and the QoC of the response time for that operation is 90%, then the user can expect that the response time of 90 out of 100 invocations will be 10 seconds. The remaining 10 invocation maybe violated, e.g. response time $>$ 10 seconds.

4. WS-QOC RUNTIME ARCHITECTURE

Various architectural building blocks of WS-QoC are discussed here, namely the elementary components needed to enable the management of the reputation through the various stages of its lifecycle. Section 4.1 gives an insight into the SLA specification, section 4.2 describes the main components of the WS-QoC framework. Section 4.3 describes the information flows and interactions between the different WS-QoC components. Section 4.4 describes our prototype implementation.

4.1 Service Level Agreements

An SLA defines mutual understandings and expectations of a service between the service provider and service consumer. The service guarantees are about what transactions need to be executed and how well they should be executed. An SLA must have the following components:

Parties: describes the participants involved in the SLA and their respective roles.

Validity Period: defines the period of time over which the SLA will be valid.

Scope: defines the services covered in the agreement.

Service Level Objectives: are the levels of service that both the users and the service providers agree on, and usually include a set of service level parameters, such as availability and execution performance. Each aspect of the service level, such as availability, will have a target level to achieve.

Service level parameters: the means by which these levels can be measured.

Typically, each Web service interacts with many other Web services, switching between roles of being a provider in some interactions and a consumer in others. Each of these interactions could potentially be governed by an SLA. The Web Service Level Agreement (WSLA) [7] project addresses service level management issues and challenges in a Web Services environment, for particular SLA specifications. The project was designed to specify SLAs in a precise and flexible manner to automate the process of monitoring and metric collections. An agreement that follows the WSLA specification compromises three sections: parties, service definitions and obligations. The parties section contains contact and technical information about the involved parties. Two types of parties are defined: signatory parties, namely service provider and service consumer, and supporting parties. The supporting parties in the context of WS-QoC are the monitoring service, the SLA deployment service and the compliance service, see section 4.2 for details about these services. The service definition section contains one or more service objects. A service object is an abstraction of a service (e.g. a WSDL operation), whose properties are relevant for defining guarantees described in the SLA. A metric within such an SLA specifies how a value is measured (by defining a measurement directive) or specifies a function to compute the parameter. The function can take other metrics and other input into account. Figure 1 is an example of the service definitions we use in our prototype implementation. The service definition contains one SLA parameter: `ProcessTimeRatio` which is composed from the `ProcessTime` (representing the time taken to process input data). WS-Agreement [4] is another specification which offers an XML language for specifying an agreement between a resource/service provider and a consumer, and a protocol for creation of an agreement using agreement templates. The main difference between WSLA and WS-Agreement is that WSLA, unlike WS-Agreement, has a concrete specification for describing how the metrics can be monitored.

```
<ServiceDefinition name="ClassifierService">
    <Operation name="classifyDataSet"
         xsi:type="WSDLSOAPOperationDescriptionType">
      <SLAParameter name="OverloadPercentage"
         type="float" unit="Percentage">
        <Metric>OverloadPercentageMetric</Metric>
    <Communication>
      <Source>Provider_A</ Source >
      <Pull>ZAuditing</Pull>
      <Push>ZAuditing</Push>
    </Communication>
      </SLAParameter>

      <Metric name="ProcessTime" type="float" unit="seconds">
        <Source></Source>
        <MeasurementDirective xsi:type="SLA_ID" resultType="float">
            <RequestURL>
                http://ProcessTimeMonitoringService/
            </RequestURL>
        </MeasurementDirective>
      </Metric>
  </Operation>
</ServiceDefinition>
```

Figure 1: The definition of the service in terms of the service parameters and their measurement.

The obligation section contains two types of obligations: a service level objective and an action guarantee. A service level objective is a guarantee that a specific SLA parameter will have a particular value within a given time period. The action guarantee is the agreement to execute a particular task within a defined situation. For example: a notification is sent if a service level objective is violated. Every obligation has an obliged party. Service level objective are typically the obligation of the service provider, and not of supporting parties. Figure 2 is an example of a service level obligation constructed from the service definition shown in Figure 1. The `ServiceLevelObjective` in this figure states that the process time must be less than 10 seconds. The `ActionGuarantee` shows that a violation notification should be sent to the SLA Deployment Service in case of a violation that occurs during the process time.

4.2 WS-QoC Components

The architecture of the WS-QoC consists of six components: the service provider, the service consumer, the monitoring

```
<Obligations>
    </ServiceLevelObjective>
    <ServiceLevelObjective name="ProcessTimeLessThan10">
      <Obliged>ServiceProvider_Y</Obliged>
      <Validity>
          <Start>2004-03-25T14:01:00.000-05:00</Start>
          <End>2004-03-25T14:02:00.000-05:00</End>
      </Validity>
      <Expression>
        <Predicate xsi:type="Less">
          <SLAParameter>ProcessTime</SLAParameter>
          <Value>10</Value>
        </Predicate>
      </Expression>
      <EvaluationEvent>NewValue</EvaluationEvent>
    </ServiceLevelObjective>

    <ActionGuarantee name="ProcessTimeNotificationGuarantee">
      <Obliged>ZMonitoringService</Obliged>
      <Expression>
        <Predicate xsi:type="Violation">
          <ServiceLevelObjective>
                ProcessTimeLessThan10
          </ServiceLevelObjective>
        </Predicate>
      </Expression>
      <EvaluationEvent>NewValue</EvaluationEvent>
      <QualifiedAction>
        <Party>ZDeploymentService</Party>
        <Action actionName="notification" xsi:type="Notification">
      <NotificationType>Violation</NotificationType>
      <SLAParameter>ProcessTime</SLAParameter>
    </Action>
      </QualifiedAction>
      <ExecutionModality>Always</ExecutionModality>
    </ActionGuarantee>
   </Obligations>
</SLA>
```

Figure 2: The obligations of the parties, referring to parameters defined in figure 1

service, the SLA deployment service and the service broker – as illustrated in Figure 3. The QoC, the SLA deployment and the monitoring services in our context are services provided by trusted third parties. The concept is similar to the issuance of digital certificates from a Certification Agency (CA) where the CA is considered as a trusted third party. It is also possible for interactions between these components to be encrypted Therefore, a message receiver trusts the content of the message as well as where the message came from.

According to this architecture, the service provider publishes an SLA-enabled Web service and sends it to the service broker for advertisement in a repository. A service consumer registers with the service broker – and finalises the SLA with the appropriate service provider. Once the service provider and consumer negotiate the relevant parameters, the SLA is deployed via an SLA deployment service. The deployment service verifies the SLA's parameters and assigns the service level objectives, defined in the SLA, to the relevant parties. The transaction then is monitored by a monitoring service to detect any violations of service level objectives. Any violation is sent to the responsible party and to the SLA deployment service which captures the violation and updates the compliance reputation of the responsible party accordingly.

4.2.1 The SLA Deployment Service (SDS)

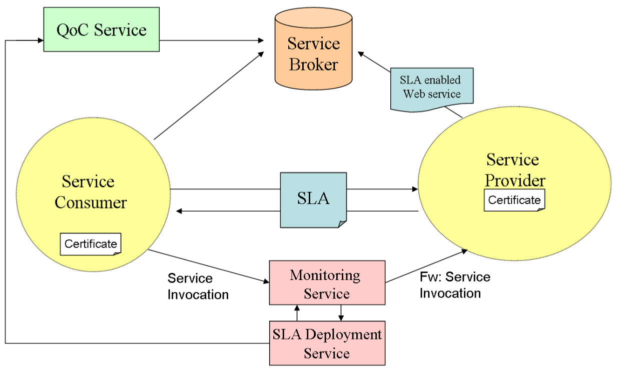


Figure 3: The Architecture of the WS-QoC Framework

The SLA Deployment Service has two primarily functions: (1) SLA activation, and (2) SLA termination. In the former case, SDS receives an SLA document signed by both signatory parties and verifies the SLA obligations of each party. The verification process confirms whether the SLA parameters can be monitored. Once, the SLA is verified, SDS distributes the obligations to the parties involved. It must be addressed that signatory parties (service provider and consumer) may do not want to share the whole SLA with thier supporting parties i.e. third parties. The SDS must extract the relevant information for each party. Thus, each party receives his own obligation only.

Once the SLA has been terminated, SDS issues a receipt and sends it to the QoC service. A specification of the receipt is proposed for the context of WS-QoC, and comprises the SLA document that was agreed upon. The service level objective section in the SLA is extended to include information about the actual values of the delivered SLA parameters. Figure 4 illustrates the specification of the extended service level objective.

```
<xsd:complexType name="ServiceLevelObjectiveType">
  <xsd:sequence>
    <xsd:element name="Obliged" type="xsd:string" />
    <xsd:element name="Expression" type="wsla:LogicExpressionType"/>
    <xsd:element name="DeliveredValue" type=" xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
```

Figure 4: The extended specification of the service Level Objective in the receipt specification

4.2.2 The Monitoring Service (MS)

The Monitoring Service (MS) is responsible for monitoring service invocations, detecting any violation of service level objectives and sending action guarantees to responsible parties.

MS is implemented as a third party. It is important to implement the MS as a third party as it is assumed that none of the signatory parties, i.e. service provider and consumer, trust each other to perform their obliged tasks correctly. Thus MS acts in a supporting role and are sponsored by either one or both signatory parties. The MS monitors the service invocation from outside the service provider's domain by probing and intercepting client invocations.

MS comprises five components: Control Management Service, Monitoring Service, Measurement Service, Evaluation Service and Notification Action Service. See figure 6 for the architecture of the Monitoring Service.

The Control Management Service receives an obligation to monitor the execution of an SLA. The main role of this service is to propagate the obligations between the subcomponents and control the interaction between them.

The Measurement Service maintains run-time information on the metrics that are part of the SLA. It measures SLA parameters such as response time or availability. Two type of measurement services are implemented: (1) SLA Measurement Service and (2) Metric Measurement Service. The Metric Measurement Service measures a particular metric such as response time. The SLA Measurement Service acts as a wrapper service or a manager for all the Metric Measurement Services for an SLA. Each SLA has its own SLA Measurement Service. See figure 5

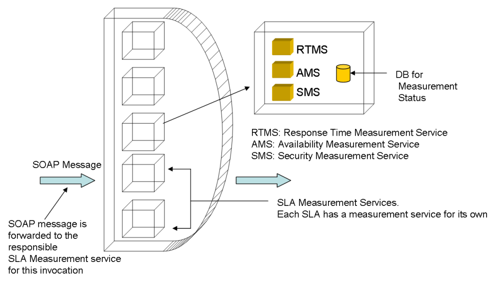

RTMS
AMS
SMS
DB for Measurement Status
RTMS: Response Time Measurement Service
AMS: Availability Measurement Service
SMS: Security Measurement Service
SOAP Message
SLA Measurement Services.
Each SLA has a measurement service for its own
SOAP message is forwarded to the responsible SLA Measurement service for this invocation

Figure 5: The Architecture of the Measurement Service

The Evaluation Service is responsible for comparing measured SLA parameters against the thresholds defined in the SLA and notify the Action Service. It obtains measured values of SLA parameters from the Measurement Service and tests them against the guarantees given in the SLA. This is done after the end of each invocation.

The Action Service is responsible for notifying the responsible parties. Each SLA obligation has ActionGuarantee clause which defined the action required in case of violation, see 2 for an example.

4.2.3 *The Service Broker (SB)*

The Service Broker in our context supports two main functions: (1) support for service discovery based on QoC and QoS metrics, and (2) support for dynamic update of QoC for services. The UDDIe [16] registry service provides the basis for the SB. UDDIe extends the UDDI registry to allow service providers to publish information about their services in a `PropertyBag` which extends the `businessService` structure found in a standard UDDI registry. UDDIe has been used primarily to publish services with their QoS information and subsequently to search for a service based on QoS properties [2]. The QoS metrics are published in the

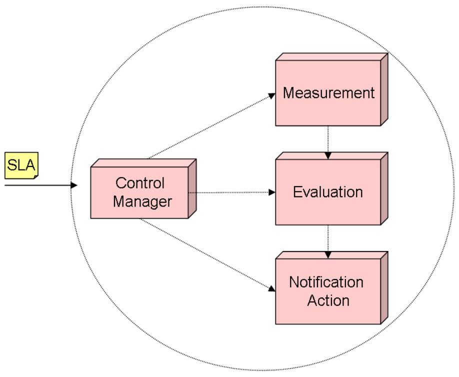


Figure 6: The Monitoring Service Architecture

`propertyBag`. We use the `propertyBag` to publish the QoC metrics for services. However, we restrict updating the QoC metrics to the Compliance Services.

4.2.4 *The QoC Service*

The QoC Service is the main component of WS-QoC. Its role is to update the QoC metrics associated with a set of services. It receives a receipt from the SLA deployment service, which includes the actual and projected values of the violated QoS metrics. For each QoS metric, the QoC service computes the difference between the predicted or suggested value, and the actual value delivered.

Hence, every QoS metric m in the SLA has a projected and an actual value. We can therefore consider the SLA to be a set: $SLA = \{m^1, ..., m^k\}$ of metrics that need to be satisfied. The projected value m_p is the value that the service consumer and provider have agreed upon, and is defined in the SLA. The actual value m_a is the value that the service provider delivers, and is measured by the monitoring service – hence: $\triangle M = (m_p - m_a)$. In the context of an SLA, therefore, we can determine $\triangle M$ for the i^{th} metric $(1 < i < k)$ – leading to:

$$normalised\ \triangle M^i = \frac{\triangle M^i}{m_a^i}$$

This normalised value allows us to ensure that we can fairly compare (within some limited bound) different metrics. $normalised \triangle M^i$ can be positive or negative. A positive value occurs when the actual value is less than the projected value, and vice versa. As not all metrics are likely to be of the same significance to a user, we can priorities each metric – and therefore also the difference observed for that metric (between the actual and the predicted values). This leads us to the concept of a weighted (by ω^i) normalised difference for a given metric, hence:

$$normalised\ \triangle M^i = \frac{\triangle M^i}{m_a^i} \times \omega^i, where\ (0 < \omega^i < 1)$$

a large difference in some metrics is considered to be more significant than others. By default, all metrics are treated in the same way ($\omega^i = 1, \forall i$) by the QoC service, and in some instances it is possible to ignore certain metrics as not being of significance in testing compliance (i.e. $\omega^i = 0$). This weighted difference can be calculated for all metrics in the SLA:

$$= \sum_{i=1}^{k} \frac{\triangle M^i}{m_a^i} \times \omega^i$$

In a strict sense, $\triangle M$ provides an indication of compliance of an SLA between a service user and provider – suggesting that the actual value provided must be *exactly* the same as that requested/predicted. However, it is possible that compliance is not intended to be exact, i.e. $\triangle M^i$ can lie within a range, or that two metrics may have a relationship that must hold true, i.e. $R(\triangle M^i, \triangle M^j)$ (for some metrics m^i and m^j) must evaluate to true for the SLA on metrics m^i and m^j to be compliant. An example of such a relationship R can be a threshold – for example:

$$R(\triangle M^i) = \begin{cases} 1 & \text{if } \triangle M^i < \textit{threshold}, \\ 0 & \text{otherwise} \end{cases}$$

this defines a Boolean relationship that can be tested with reference to a threshold. Hence, the function R here indicates that the difference between the actual and predicted value must not exceed a threshold, for that particular metric to be considered compliant. There are a variety of other relationships R that may also be defined – with such functions taking one or more arguments. Such an approach provides greater flexibility when comparing metrics within an SLA with measured values. Using this approach, we can therefore deduce that compliance for metric $m^i = R(.)$. One can now compute the compliance for an entire SLA as:

$$\textit{compliance} = \frac{1}{k} \sum_{i=1}^{k} R^i$$

which assumes that there are k metrics in the SLA (as indicated above). All of the definitions above assume a single transaction – i.e. there has been a single interaction between the service provider and the consumer. One can therefore generalise to n transactions, where compliance is now tested across a complete set of interactions between a user and provider. Hence, the normalised weighted difference for the entire SLA is:

$$= \sum_{j=1}^{n} \sum_{i=1}^{k} \omega^{(i,j)} \frac{\triangle M^{(i,j)}}{m_a^{(i,j)}}$$

and using a similar argument, the compliance for the entire SLA can be calculated as:

$$= \frac{1}{k \times n} \sum_{j=1}^{n} \sum_{i=1}^{k} R^{(i,j)}$$

4.3 Interactions between WS-QoC components

WS-QoC lifecycle consists of six distinct stages. We assume that the SLA is defined for a Web service (the interface for which is defined in WSDL), which is running within the service providers domain. The stages and the components that implements the functionality needed during the various stages are as follows:

Stage 1: Service description, advertisement and discovery based on QoS and QoC

In the context of our framework, service providers must explicitly publish their services with pre-defined QoS metrics. In the WSDL specification document, there is no explicit provision for associating QoS metrics with the service definition. Therefore, we extend the WSDL document and include an addition tag `ServiceData` similar to the `ServiceDataElement` tag in the Open Grid Services Architecture (OGSA) [8] specification. The `ServiceData` tag is utilised to encode QoS parameters associated with a service. The `ServiceData` tag is mapped to the `propertyBag` extension in the UDDIe [16] registry service, which eventually makes this QoS metrics searchable. We have developed a service publishing tool wizard that automates the publishing process by taking in two WSDL-based documents: i) service interface definition, and ii) service implementation and publishes the service in the UDDIe. The tool helps service providers to include QoS attributes associated with a service in the service interface definition document, and publish the service in the UDDIe registry – without having to write any XML code. We have also implemented a tool that automates the discovery process as illustrated in Figure 7. The tool asks the user to fill in the QoS and QoC requirements of the required service. The tool then inquiries the UDDIe registry which returns the list of matching services. Once the results are returned from the registry, the tool reasons about the QoS and QoC information for services and returns the best selected service to the service requester.

Stage 2: SLA Negotiation and Establishment

The service information published in the service broker constitutes an offer being made to the user. In this stage, the service requester retrieves the SLA metrics from the service offering, combines them into various SLA parameters and obligations and sends them to the service provider. The service provider in turn either accepts or rejects the request. In the former case, the service provider may accept all the obligations or negotiates with the service requester on varying some of the obligations. The outcome of the negotiation process is a single SLA document comprising the SLA parameters and obligations of all the involved signatory and supporting parties. Once the SLA has been established, it is sent for deployment to the deployment service.

Stage 3: SLA Deployment

The deployment process involves two steps. In the first step,

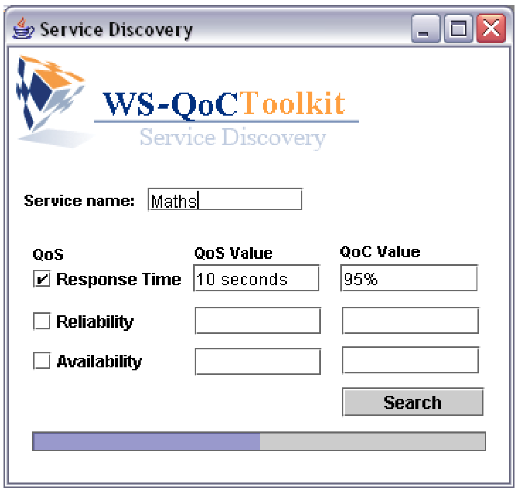


Figure 7: Discovering Services based on QoS and QoC metrics

the SLA parameters and obligations are validated. Typically this validation process would check if the SLA parameters are meaningful and can be measured. In the second step, the deployment service sends the obligations to the relevant parties.

Stage 4: SLA Monitoring

In this stage, the service level objectives are monitored. Any violation is sent to the SLA deployment service.

Stage 5: SLA Termination

The SLA may specify the conditions under which it may be terminated or the penalties a party will incur by breaking one or more SLA clauses. Negotiations for terminating an SLA may be carried out between the parties in the same way as the SLA establishment was achieved. Alternatively, an expiration date may be specified in the SLA.

Stage 6: QoC computation

Once the SLA has been terminated, the deployment service sends a receipt which includes the actual and projected values of the violated QoS metric(s). The QoC service then updates the QoC metrics of the considered service in service broker.

4.4 Prototype Implementation

WS-QoC is implemented on RedHat Linux 7.2. The programming tools are: Java2 SDK Version 1.4.0, UDDIe registry service, WSLA4J and the Tomcat application server. A monitoring service which supports some of the functions outlined in section 4.2.2 has also been implemented. The publicly available API (UDDI4J) is used to facilitate the interaction between the deployment service and UDDIe. In the following subsections we will discuss the implementation details of measuring the response time and the availability of the service.

4.4.1 Measuring Response Time

The monitoring service measures the response time metric in an SLA. Figure 8 illustrates a GUI of the implementation. The figure shows the result of monitoring a service operation. The response time for this operation was agreed upon to be less than or equal to 10 seconds for 15 invocations. The figure shows that in invocation number 9 and 10 there was a violation.

Response time is a common and universal measure of performance. For Web services, it can be defined as the guaranteed average time required to complete a service request. Given an operation op for a service s, the response time can be broken down into two major components: delay time and process time. Delay time (DT) refers to the time needed to transmit data to the service. Process time (PT) is the time a Web service takes to process and execute an operation op. The response time for an operation op for a service s is therefore computed as follows:

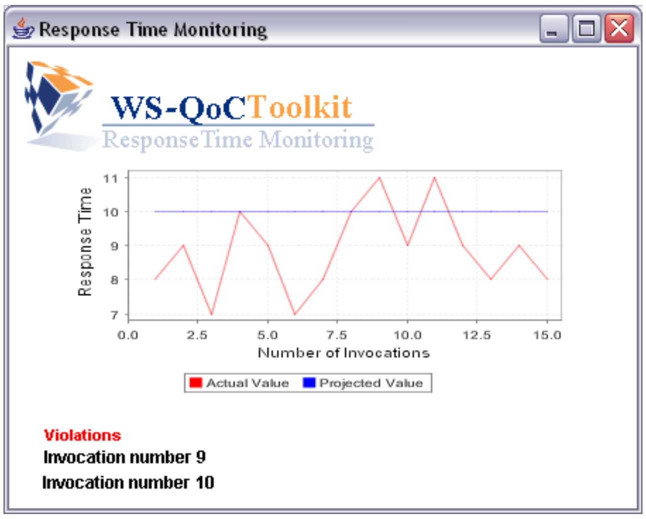
$$T(op, s) = DT(op, s) + PT(op, s)$$

The process time can be further broken down into queuing delay, setup delay and execution time. Queuing delay is the time a service request spends waiting on the service provider side, before the request is selected for processing. Setup delay is the time needed to set up the Web service. Setup activities may correspond to authorisation and authentication processes. The execution time is the actual execution time of the operation. In our framework, we only measure the PT of a Web service.

The response time monitoring service is implemented as a Web service that monitors the service invocation by probing and intercepting message follows between the client and service provider. As discussed in section 4.4.2 Measurement, Evaluation and Notification Action services are implemented as the core components of the monitoring service. A Metric Measurement Service for the response time is implemented to capture the response time for service invocation. Figure 9 depicts the message path on the service provider domain before it reaches the Apache Axis server.

A message arrives (in some protocol-specific manner) at a Transport Listener. In our case, the Listener is an HTTP Java Servlet. The Listener packages the protocol-specific data into a Message object `(org.apache.axis.Message)`, and puts the Message into a `MessageContext`. Once the `MessageContext` is ready to go, the Listener hands it to the Monitoring Service. The Monitoring Service's first job is to send the Message to the SLA Measurement Service that is responsible on monitoring the incoming message. Once the Measurement Service receives the Message it forwards it to the Metric Measurement Service responsible for measuring the response time. The later service records the current time and hands the message to the service provider. When the response is returned from the service provider, the service measures the response time as:

Response Time = the time when the response is returned from service provider - the time when the message was sent to Metric service.



Violations
Invocation number 9
Invocation number 10

Figure 8: Response Time Monitoring Service

The Measurement Service keeps a record for the these measurement in a database. The Evaluation service in turn query Measurement service to test for violation. In case of violation, the Action Service in notified as it was discussed in section 4.2.2.

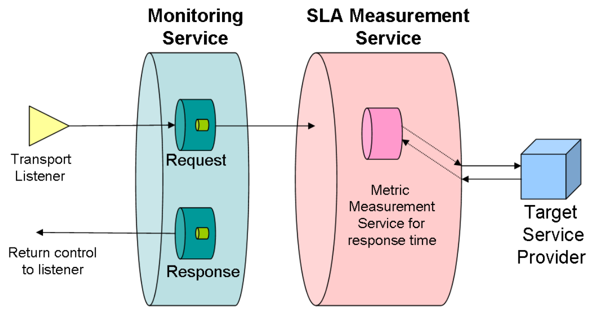


Figure 9: The Architecture of the Response Time Monitoring Service

4.4.2 *Measuring Availability*

The monitoring service also measures the availability metric in an SLA. Measuring Service availability is a straight forward practice. There are many times when the service provider fails to response to a request message. We assume that the service provider and requestor are already bonded by an SLA agreement to deliver a particular service. The failure might be due: (1) the service is off-line, or (2) high system workload or a system fault (although we are aware that there are other reasons why a service request can fail, at present we only focus on these as they are the most common reasons a detailed analysis can be found in [10]). Each time the service provider fails to response to a request message, the monitoring service captures the failure as an SLA violation.

5. CONCLUSION

The rapid growth of Web services indicates that service requesters must consider many factors when selecting the "right" service. In this paper, we introduced a new aspect of quality, namely the Quality of Compliance (QoC). Managing QoC provides a mechanism for assessing the level of compliance of the service provider to some pre-agreed contract, and therefore gives an indication of the actual QoS values that a service provider has been able to deliver. The concept of QoC stems from the fact that there has been no practice of recording the achieved service levels once a transaction has been completed. Doing so gives an insight into the providers past performance by providing necessary data to progressively assess the compliance levels over a range of past transactions. We also presented in this paper a framework (WS-QoC) and prototype implementation for integrating the QoC into Web services.

Our framework consists of four components implemented as Web services: the monitoring service, the SLA deployment service, the QoC service and the service broker. In this framework, the service provider publishes an SLA-enabled Web service and sends it to the service broker for advertisement in a repository. A service requester registers with the service broker, searches for QoS and QoC aware services and finalises the SLA with the appropriate service provider. After the provider and the consumer negotiate the relevant parameters, the SLA is deployed by the service (via the deployment service). The deployment service verifies the SLA's parameters and assigns service level objectives to the relevant parties. The transaction then is monitored by the monitoring service to detect any violations of service level objectives. Any violation is sent to the QoC service which captures the violation and updates the QoC metrics of the service in the service broker.

6. REFERENCES

[1] A. Abdul-Rahman and S. Hailes. Using recommendations for managing trust in distributed systems. *Proceedings IEEE Malaysia International Conference on Communication*, 1997.

[2] R. Al-Ali, O. Rana, D. Walker, S. Jha, and S. Sohail. G-qosm: Grid service discovery using qos properties. *Computing and Informatics Journal, Special Issue on Grid Computing*, 21(4):363–382, 2002.

[3] R. J. Al-Ali, K. Amin, G. von Laszewski, O. F. Rana, D. W. Walker, M. Hategan, and N. Zaluzec. Analysis and provision of qos for distributed grid applications. *Journal of Grid Computing (Kluwer)*, (to appear) 2004.

[4] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. Web services agreement specification. *http://www.gridforum.org/Meetings/GGF11/Documents/draft-ggf-graap-agreement.pdf*, 2004.

[5] F. Azzedin and M. Maheswaran. Evolving and managing trust in grid computing systems. *Proceedings IEEE Canadian Conference on Electrical and Computer Engineering*, 2002.

[6] P. Bhoj, S. Singhal, and S. Chutani. Sla management in federated environments. *Hewlett-Packard Labs Technical Report HPL-98-203*, 1998.

[7] A. Dan, D. Davis, R. Kearney, A. Keller, R. King, D. Kuebler, H. Ludwig, M. Polan, M. Spreitzer, and A. Youssef. Web services on demand: Wsla-driven automated management. *IBM Systems Journal*, March 2004.

[8] I. Foster and C. Kesselman. The physiology of the grid:an open grid services architecture for distributed systems integration. *Techinical Report*, Jan 2002.

[9] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. *Proceedings of the Twelfth International World Wide Web Conference*, 2003.

[10] M. Klein and C. Dellarocas. A knowledge-based approach to handling exceptions in workflow systems. *Journal of Computer Supported Cooperative Work*, 9(Issue 3-4), August 2000.

[11] A. Mani and A. Nagarajan. Understanding quality of service for web services. *http://www-106.ibm.com/developerworks/webservices/library/ws-quality.html*, 2002.

[12] P. Obreiter. Case for evidence-aware distributed reputation systems - overcoming the limitations of plausibility considerations. *Proceedings of the 4th International Workshop on Cooperative Information Agents IV, The Future of Information Agents in Cyberspace*, 2000.

[13] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. *Stanford Digital Library Technologies Project*, 1998.

[14] S. Ran. A model for web services discovery with qos. *ACM SIGecom Exchanges*, 4(1), March 2003.

[15] A. Sahai, A. Durante, and V. Machiraju. Towards automated sla management for web services. *Hewlett-Packard Labs Technical Report HPL-2001-310 (R.1)*, 2002.

[16] A. ShaikhAli, O. Rana, R. Al-Ali, and D. Walker. Uddie: An extended registry for web services. *Proceedings of the Service Oriented Computing: Models, Architectures and Applications, SAINT-2003 IEEE Computer Society Press.*, 2003.

[17] M. Tian, A. Gramm, T. Naumowicz, H. Ritter, and J. Schiller. A concept for qos integration in web services. *1st Web Services Quality Workshop (WQW 2003), in conjunction with 4th International Conference on Web Information Systems Engineering (WISE 2003)*, 2003.

[18] A. Vivacqua. Agents for expertise location. *Proceedings of the 1999 AAAI Spring Symposium on Intelligent Agents in Cyberspace*, 1999.

[19] B. Yu and M. P. Singh. A social mechanism of reputation management in electronic communities. *Proceedings of the Second International Conference on Trust Management (iTrust'04)*, 2004.

[20] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Z. Sheng. Quality driven web services composition. *Proceedings of the twelfth international conference on World Wide Web*, 2003.

Distribution Concerns in Service-Oriented Modelling

N.Aoumeur, J.L.Fiadeiro, C.Oliveira
Department of Computer Science
University of Leicester
Leicester LE1 7RH, UK

{na80,jwf4,co49}@leicester.ac.uk

ABSTRACT

Service-oriented development offers a novel architectural approach that addresses crucial characteristics of modern business process development such as dynamic evolution, intra- and inter-enterprise cooperation, and distribution/mobility. In previous papers, we have shown how the mechanisms that regulate the relationships, functioning and cooperation of business activities in such architectural models can be externalised from business rules in terms of connectors that can be superposed dynamically on stable core business entities. That is to say, we focused on what, in the literature, has been called the "service composition layer" of service-oriented architectures or, for short, their "composition logic". Our emphasis in this paper is on the distribution aspects: we show how a corresponding "distribution logic" can be defined in terms of another set of architectural primitives that address the way business rules depend on "locations". These primitives address what are sometimes called "business channels" (ATMs, PDAs, Pay-TV, Internet, inter alia) as offered in typical contemporary ICT-infrastructures with substantial added-value to business processes. We argue that interacting (core) business entities located at or endowed with such ICT capabilities should be modelled in a way that separates the composition from the distribution logic so that business interactions can be understood and evolved in a location-transparent way. Our approach is based on a mathematical model that we have recently developed for modelling context-aware interactions. An example from banking is used for illustrating its applicability.

Categories and Subject Descriptors

D.1.3 [**Concurrent Programming**]: Distributed Programming, Parallel programming. D.2.11 [**Software Architectures**]: Languages – *connectors;* F.1.2 [**Modes of Computation**]: Interactive and reactive computation.

General Terms

Design, Languages, Verification.

Keywords

Evolution, location-awareness, rule-based business modelling, service composition and coordination, software architectures

1. INTRODUCTION

Modern business processes are becoming more and more complex, reflecting the increasing dependency of the economy, and the functioning of the society as a whole, on intricate and volatile intra- and inter- organisational cooperation. On the other hand, business operations are relying more and more on day-to-day advances in Information and (wired/wireless) Communication Technology (ICT). In order to remain competitive, respond to market pressure and attract more customers, companies are pressed to provide ever more sophisticated added-value services. For instance, banks are continuously creating new services or updating existing ones to match the expectations and profiles of their customers, while at the same time supporting more and more advanced channels for day-to-day banking such as ATM, Internet, PDA, Pay-TV, inter alia [23].

This tension between complexity and agility is raising new challenges on the way software needs to support business information systems. It is clear that these challenges transcend by far the capabilities of the software engineering techniques that have been traditionally used for business process development. This is why most business designers are looking for new solutions around workflows [1] and, more recently, web services [34]. As a result, significant standards, techniques and models have been advanced in both directions for modelling and enacting business processes.

However, we argue that the *operational* character of these approaches (even when supported by mathematical models like BizTalk [8,24]) makes it very hard to tackle all the above features adequately. Although it is widely accepted that abstraction and rigor are the preponderant means for tackling levels of multi-dimensional complexity, addressing these requirements equally and coherently, as their nature and expected added-value determine, requires a more declarative approach and semantic modelling primitives that work at a level of abstraction in which the different dimensions can be integrated and reasoned about.

More specifically, on the one hand, current standards lack rich mechanisms like service negotiation, contracting and service communication and coordination as required for flexibility and *dynamic* adaptation and evolution [7] in cross-organisational processes. In addition, despite some progress in semi-automatic derivation of service-oriented business processes from informal business rules [26], the relationship between business rules, their evolution and web-services in general remains largely unexplored.

On the other hand, proposals based on Web services experience serious difficulties in addressing location-awareness as an essential business concern for dealing with multi-channels provided by present day's technology. Web services can be programmed in ways that respond to the need for businesses to operate in different platforms and through different channels (say, banking at an ATM, across the internet, or through a PDA/mobile phone), but Service Description Languages do not provide abstraction mechanisms for modelling the underlying *distribution logic* and the way it adheres and enforces given *business policies.*

The purpose of this paper is to put forward a set of primitives through which distribution concerns can be addressed in service-oriented business modelling. We do so by extending the approach that we have put forward in [6] for addressing the composition logic, i.e. "the way composite services can be constructed for defining processes or workflows that interact with sets of Web Services to achieve certain goals" [11,32].

In section two, we justify the use of a rule-based architectural approach for modelling both the composition and the distribution logic of services, discuss the main assumptions that we make on the way service-oriented development applies to business processes, and present the running example – a simplified banking system. In section three, the coordination primitives that address the composition logic are reviewed and illustrated using the example. In section four, location primitives are presented as the building blocks for the envisaged distribution logic, and illustrated using the same example. In section 5, we present an architecture for modelling and evolving agile and dynamic business processes based on coordination and distribution.

2. MOTIVATION

In this section, we justify why and how we are bringing together concepts and techniques from service-oriented development, rule-based business modelling, software architecture, and context-aware computing.

A rich set of specifications is currently available for software development over service-oriented architectures that include the Business Process Execution Language for Web Services (BPEL4WS or BPEL for short) [9], WS-Coordination [35] and WS-Transaction [36], *inter alia*. A so-called BPEL composition is a business process or workflow that interacts with a fixed set of Web services to achieve a certain goal. A business process is taken as a series of activities involving a given set of partners connected according to given data and control flow requirements. For instance, a *banking process* can be taken to consist of several activities, including specifically:

- Customer identification and authentication.
- Customer execution of banking transactions (deposits, withdrawals, loans, mortgages, etc).
- Customer exit.

Web Services are "self-contained, modular applications that can be described, published, located, and invoked over a network, generally the Web" [34]. They are capable not only of performing business activities on their own, but also to take part in higher-order business transactions by engaging in more or less complex interactions with other Web services.

This approach offers a significant number of advantages. For instance, by being platform-neutral, Web services support the definition of business processes by using existing elementary or complex services, possibly offered by different service providers or extracted from so-called legacy systems. However, even applications developed on the basis of BPEL are still some way from addressing the challenges raised by the need to tackle complexity and agility as identified in the introduction. One of the reasons is that BPEL-style applications are rather unstructured and static. For instance, services are composed in a rather ad hoc and unprincipled manner by simply combining their operations and input and output messages. This makes business processes difficult to evolve. If the business rules under which the process operates change or need to be adjusted, the workflow will have to be revised and additional or modified service interfaces may have to be used for the interconnections.

Recent investigations in business process modelling are shifting the emphasis towards more abstraction through business rule-driven approaches [15,29]. Business rules are understood as "projections of organisations' constraints and declarations of (internal/external) policy/conditions that must be satisfied for doing business" [33]. They specify actions to be taken on the occurrence of particular events, including "state of being" changes concerning individuals, infrastructure, consumables, informational resources, and business activities in general.

Rule-driven approaches offer a number of advantages that are crucial for coping with dynamically evolving complex business processes. They support the specification of business models *independently* of the specific processes that happen to be running at any one instant. They focus on more primary requirements and address business domain descriptions in a declarative rather than operational way. For all these reasons, they are generally more apt to support *evolution*.

The exploitation of these potentials for achieving new degrees of dynamism and abstraction in Web Services composition remains largely unexplored. An exception is the recent work by Papazoglou et al. [24]. In this approach, starting from a very general specification, the composition is scheduled, constructed and finally executed with the assistance of business rules judiciously classified in a repository. Besides basic elements such as events, conditions, and messages, this classification includes rules dealing with the activity flows, the data required for their composition and the constraints to be respected. The direct construction and subsequent execution of the composition from the business rules is performed in terms of XML-like descriptions. However, the approach does not address the distribution dimension.

Our contribution follows in this path and aims to enhance the potential of service-oriented architectures by developing semantic primitives that raise the level of abstraction and capture rule-based business modelling. In the approach that we have in mind, each business activity is a dynamic entity that is put together, at run-time, from a number of self-contained applications (services) that need to be located and invoked over a distribution network. The way these services are brought together and invoked, what is sometimes called "orchestration", must follow given business rules as set by the organisation. For instance, it is clear that a withdrawal activity is subject to the business rules that apply to the specific customer and account involved as business entities. Depending on the nature of the account and of the customer, certain constraints may apply that determine if, for a given amount, the withdrawal is authorized and, if so, what operations of the bank itself need to be executed.

More specifically, our approach aims to capture business rules at an interaction level so that dynamic adaptation of services and cross-organisational service cooperation can be intrinsically and explicitly supported (composition logic). For this purpose, we adopt techniques akin to those that have been developed for Software Architecture [3]. We propose to capture as a *connector* any business rule dealing with intra- and inter-organisation cooperation. On the one hand, as modelling primitives, architectural connectors can be made to describe business service compositions in a declarative way as shown through the rule-based approach proposed in [5]. On the other hand, as shown in [22] architectural approaches support dynamic evolution as required for agility and reconfigurability.

In our approach, the mechanisms that are required for regulating the relationships, functioning and cooperation of services are externalised from business rules in terms of semantic primitives that we call coordination laws. These describe composition mechanisms in terms of event-condition-action (ECA) rules that can be superposed dynamically on stable core business entities. Superposition [16] is non-intrusive on the code that implements the services. Therefore, business architectures can be dynamically evolved, as volatile business rules change or new cross-organisational links come into force, while ensuring compliance to core business invariants.

However, business activities depend on business channels and networks in ways that are orthogonal to the interactions that business relationships impose. For instance, depending on the location where the banking process is requested, identification and authentication can consist of:

(1) A simple "hello" when the request is made by the customer in person at the desk of the branch where the account has been held for 20 years.

(2) The presentation of a personal identity document if the clerk has only recently joined that branch or if the customer at a different branch makes the request.

(3) A complex transaction involving debit cards and codes if the request is made at an ATM (not necessarily by the customer).

(4) A collection of passwords, security codes and pre-determined personal questions if the request is made through the Internet (again, not necessarily by the customer).

In terms of a service-oriented architecture, this means that the way composite services need to be constructed should obey not only a composition logic derived from coordination concerns, but also a distribution logic derived from location concerns. Indeed, location-awareness is common to business channels (e.g. ATM, Branch, Pay-TV), mobile devices (e.g. PDA), internet-based facilities/software, and sensors, inter alia. The presence and quality of communication with other partners as well as the ability to migrate or move to other locations are among the crucial features that need to be taken into account at the level of this distribution logic.

Notice that, by location, we do not mean necessarily the space of addresses typically used in the Web. In the literature, service-oriented modelling is almost always instantiated to "Web Services", i.e. "software that can process XML documents it receives through some combination of transport and application protocols" [31]. Such services need to be located and invoked over the Web using addresses and referencing mechanisms that identify where services can be found using a given protocol like TCP or HTTP.

We have already motivated that this is a rather low-level view of what can be called the "service-oriented paradigm", which we would like to explore from the point of view of business process modelling in the architectural approach that we motivated. In particular, we would like to distance ourselves from both the XML-centred view of information exchange, and the Web-oriented notions of location and reference protocols. Our proposal is to work on a space in which locations correspond to business entities and channels organised according to a given business domain. Therefore, we do not work with a fixed notion of location at all. We propose that, as part of business modelling, the notion of location and distribution network that best applies to the business domain be specified in abstract terms through data sorts and operations.

To the best of our knowledge there is no *conceptual modelling* approach that addresses location-awareness in business processes in the sense that we have motivated, except for the work in [2], one of our main sources of inspiration. This work invokes the notion of "channel" for addressing location-awareness. It is, altogether, rather "operational", not as declarative as we wish ours to be, because it uses state machines as a modelling tool. It does not cope with the evolutionary side either, and it has not been integrated within an architectural approach that provides explicit connectors that can handle location-dependency aspects.

This is why, in what concerns the distribution logic that captures the dependency on the business channels and networks, we propose an approach based on explicit connectors that we call location laws. As with coordination laws, these connectors can be superposed dynamically and evolved independently of the other business aspects, allowing systems to self-adapt or be adapted to changes that occur at the distribution level without interfering with the core business policies.

The semantics of our distribution logic builds on our recent work around CommUnity, a formal approach that we have been developing for architectural description [14]. CommUnity includes primitives that capture distribution and mobility aspects [4,19]. The whole approach has a mathematical semantics defined over Category Theory [13]. We borrow in particular the notion of space of mobility (location structure) and corresponding contexts with the "be-in-touch" and "reach" relationships as preconditions for communication and mobility. These ingredients are then combined in a new format for condition-action rules that model the way service composition depends on the properties of the current context.

3. COORDINATION CONCERNS

Coordination primitives, as we have been promoting in recent work [5], provide a clean separation between the modelling of the computations performed by stable core entities on the running business configuration to ensure the functionality of basic business services, and the mechanisms that reflect how the (intra- or cross-organisational) interactions between these business services should be coordinated according to given business rules.

The emphasis is, therefore, on the aspects that subsume what in the literature has become known as the "Service Composition Layer" of Service-Oriented Architectures [27], i.e. the level at which business processes can be put together from elementary services. We aim for the level at which so-called business protocols and processes [11] are addressed. What we have in mind is the definition of processes or workflows that interact with sets of Web services to achieve certain goals in terms of abstract service descriptions, separated from specific deployments. In our approach, such interactions are captured using the concepts of coordination laws and interfaces. In terms of architecture description languages, these correspond to connector types and roles. In terms of business modelling, they capture business rules that regulate and compose required and provided services by the core entities that instantiate the roles.

This view addresses the emphasis put by BPEL [9] on the definition of service compositions in terms of processes that interact with partners that are external to the composition itself and identified only in terms of abstract interfaces. Indeed, it is particularly important that we are able to separate the definition of the "composition logic" from the run-time composition of specific services as part of a process that is being executed to fulfil a specific business goal. We address the former in terms of "coordination laws" that capture the business rules according to which complex business activities are put together from more basic services. The purpose of this section is to focus on the coordination model that we adopt for composing abstract services according to business rules.

In fact, in our approach, we go one step further and assign partners not to the business process as a whole but to the activities that are performed as part of the process. This recognises the fact that the partners involved in one activity may be different from those in another activity within the same process. Moreover, it may not be possible to pre-determine which partners will become involved in a given activity as this may depend on what has happened in the process so far.

The abstract description of the services that are partners in a given business is made in terms of what in [5] we called *coordination interface*. For instance, as a business activity, a withdrawal involves both an account and a customer regardless of the way the withdrawal is requested, if by the customer proper or anyone else. The purpose of the identification activity is, precisely, to determine the business entity that is involved in the business activity. Hence, in the case of a withdrawal, two coordination interfaces are required: one catering for the account service through which the debit needs to be performed, and the other for the customer service that is invoked as a result of the identification and authentication activity.

Note that these are "business" partners, not software components that offer operations as in object-oriented approaches. We fully support the distinction made in [31] between Web-services and distributed objects. In this paper, we are in no way concerned with the way services are programmed and deployed. For us, an account is not a software component that instantiates an object class. An account is understood as a business service, a unit of organisation around which a number of operations are grouped together to fulfil certain goals.

Such business partners are not units of execution either. A customer does not perform a withdrawal by calling the account to execute a debit. It is the composition logic, as captured by a coordination law, that dictates that a debit, as an operation of the account service, needs to be invoked whenever a customer issues a request for a withdrawal, say at an ATM through some combination of keys and buttons. The debit is to be located according to the account as a business entity, not as a software component that stores the code of the debit operation.

The trigger/reaction mode of coordination that our approach supports requires that each coordination interface identifies which *events* produced at execution time are required to be detected as triggers for the process to react, and which *operations* must be made available for the reaction to superpose the required effects. Notice that this separation is supported, for instance, in BPEL processes, by distinguishing between different kinds of actions (e.g. synchronous request/response or asynchronous one-way operation) that implement interactions among the process and its partners. Indeed, in BPEL, this separation occurs at a lower level of abstraction and has to be set in a pre-defined, static way. The same applies to the identification of the exchange of messages that such modes of interaction may require between the partners involved: in WSDL, each operation/event in our sense is a sequence of input and output messages.

The two composition interfaces that we have in mind can be described as follows:

```
coordination interface CW-CI
partner type   CUSTOMER
operations     owns(a:ACCOUNT):Boolean
events         withdraw(n:money; a:ACCOUNT)
end interface
```

```
coordination interface AW-CI
partner type  ACCOUNT
operations
    balance():money
    debit(a:money)  post balance() = old balance()-a
end interface
```

Each interface identifies the type of the partner that it models. A coordination interface does not identify a specific instance of this type, just the operations and events that partner instances are required to make available. Notice how the properties of the operations that are required are specified in an abstract way in terms of pre- and post-conditions.

This type should be specified in terms of a sort of business identities and functions that can relate the partner to other business entities as required by the application domain. For instance, the sort *ACCOUNT* should be provided with a function *bank* of type *BANK* identifying the bank in which it resides, again as business entity, not as a software component. To be more precise, as discussed in section 5, *a:ACCOUNT* may identify a service that is running as part of a bigger service *bank(a):BANK*. That is, we are not necessarily committed to creating a new independent service upon instantiation of a coordination interface: we may bind the interface to a running service that will take the instance of the interface as a sub-service. In this way, we may cater for situations in which the bank, as an organisation, runs a separate service for each account, one single (complex) service for all accounts, one single (huge) service for the whole bank, and so on.

Another important requirement for the intended composition logic is that the activity, as a composite service itself, should be described only on the basis of the interfaces and the data and control flow aspects that the coordination mechanisms put in place to ensure the underlying business goal. This is what, in BPEL, would be called the "state and logic" necessary for coordinating the interactions between the process and the partners. This "composition logic" can be described in terms of what we call a coordination law [5]:

```
coordination law SW-CL
partners    acco:AW-CI; cust:CW-CI
rules
when  cust.withdraw(n,acco)
      with acco.balance() ≥ n &
           cust.owns(acco)
      do   acco.debit(n)
end law
```

Besides identifying the coordination interfaces, a coordination law specifies the rules that define the behaviour of the service. Such coordination rules are of the form:

```
when  event
      with condition
      do   set of operation invocations
```

Each coordination rule identifies, under the "when" clause, a trigger to which the contracts that instantiate the law will react – a request by the customer for a withdrawal in the case at hand. The trigger can be just an event observed directly over one of the partners or a more complex condition built from one or more events. Under the "with" clause, we include conditions (guards) that should be observed for the reaction to be performed. If any of the conditions fails, the reaction is not performed and the occurrence of the trigger fails. Failure is handled through whatever mechanisms are provided by the language used for deployment. See [9] for explicit handling of faults within BPEL.

The reaction to be performed by the composite service is identified under the "do" clause as a set of elementary activities. This set may include calls to operations provided by one or more of the partners as well as actions that are internal to the "composition logic" of the business activity itself. The whole interaction is handled as a single transaction, i.e. it consists of an atomic event in the sense that the trigger reports a success only if all the actions identified in the reaction execute successfully and the conditions identified under the "with" clause are satisfied. Details on transaction protocols for web-service interactions can be found in [36].

In what concerns the language in which the reactions are defined, we normally use an abstract notation for defining the synchronisation set as above. This is important for bringing to a more abstract modelling level the definitions of business processes that recent languages for "orchestration" like BizTalk [8] promote, in terms of algebras and models for concurrency. Our opinion and experience is that the architectural modelling level at which we promote the representation of business interactions makes it easier to bridge the gap from the more organisational high-level goals and policies that dictate how business should be run to the choice of particular control and synchronisation structures that can make specific processes run.

The externalisation of this composition logic in a coordination law is decisive for supporting the required agility in terms of dynamic business evolution. The fact that the conditions on which an account may be debited by its owners are not hard-coded in the operations made available by the account, make it possible for these conditions to be changed without interfering with the deployment of these services. For instance, in order to offer a VIP-withdrawal in which a given credit limit is allowed, we just have to change the composition logic as modelled by the coordination rule; the basic debit operation does not need to be changed.

```
coordination law VIPW-CL
partners       acco:AW-CI; cust:cCW-CI
rules
when  cust.withdraw(n,acco)
      with acco.balance()+cust.credit()≥ n  &
           cust.owns(acco)
      do   if acco.balance()≥ n
              then acco.debit(n)
              else acco.debit(1.01*n)
end law
```

Notice that a different partner is now required to play the role of the customer: we need a service that offers an operation for obtaining the credit limit currently assigned to the customer:

```
coordination interface cCW-CI
partner type   CUSTOMER
operations     owns(a:ACCOUNT):Boolean
               credit():money
events         withdraw(n:money; a:ACCOUNT)
end interface
```

Coordination interfaces can be hierarchically organised so as to facilitate location and binding of specific concrete services. We leave such matters to a subsequent paper.

4. LOCATION CONCERNS

This section puts forward the concepts and constructions that we are developing for addressing location-awareness in service-oriented business modelling. As emphasized in the introduction, our purpose is to provide elements for a "distribution logic" that can capture the way service composition needs to take into account properties of the underlying business channels and communication infrastructure. Just like coordination mechanisms that separate service functionality from the "composition logic", which we illustrated in the previous section, we want to define location primitives that can externalise the way business activities depend on properties of the distribution topology over which services are composed. The properties that we address in the paper are:

(1) The communication status, i.e. the presence, absence, or quality of the communication link between locations where given services are executing but require data to be exchanged and synchronisation protocols to be observed as part of the composition logic.

(2) The ability to continue the execution of an activity at another location, which requires the new location to be reachable from the present one for the execution context to be moved.

For this purpose, we capitalise on the work developed around CommUnity [19]. Although, for simplicity, we will not address this specific aspect in depth, the space of locations can be defined by the user as an abstract data type with a sort *loc* and functions that capture the properties of the notion of location that are suitable for the application domain at hand. This is because, typically, different kinds of applications require different notions of location. When a specific notion of location is fixed, as for instance in Ambients [10], modelling a different space of mobility requires the encoding of a different notion of location, which can be cumbersome and interfere with other aspects. Two observables capture location awareness as discussed above: communication is handled through BT:*set(Loc)* and movement/reachability through REACH:*Loc×Loc*.

As we did for the composition logic through coordination laws, location laws are the means through which we model the distribution logic of a given business domain. Whereas coordination laws interconnect partners that are meaningful for the underlying composition logic, e.g. customers and accounts in the case of the withdrawal, the partners involved in location laws derive from the distribution logic and, therefore reflect business channels like ATMs, bank branches, etc.

That is to say, for the distribution logic of a withdrawal, what is important is not if the customer has a VIP-contract with the account, but whether the ATM at which the request for the withdrawal is made has enough cash in store and is in touch with the branch in which the account is held. The composition logic will determine whether the withdrawal can proceed according to the relationship that exists between the customer and the account, whereas the distribution law will determine how much money can be given according to the context in which the transaction is being made (cash available at the ATM and status of the communication between the ATM and the branch).

Just like with coordination laws, locations laws are associated with business activities within a process, not with the process as a whole. This is because we want to allow for business entities to change location during the process. For instance, we may well envisage an instantiation of the banking process in which the customer is a mobile entity that starts the process and performs some activities through a PDA while driving to the bank where, upon arrival, he continues by performing other activities until he eventually finishes the process over the internet in his office where he needed to retrieve information that he was lacking at the bank. The modelling of this kind of mobility within a business process is still under active research and will not be further discussed in the paper. See [4,19] for the mathematical domain over which we are defining these aspects and early insights on how to use them.

Requirements on the location of the distribution partner is an obligatory feature in every location interface. These requirements consist in the definition of the type of the location as a subtype of *loc*, including any relevant functions and properties. For instance, if a location is required to handle high-precision calculations, its type needs to be such that, upon instantiation, service operations are executed on hardware that complies with the required properties. Security requirements may be reflected in other properties and functions on the data that is transmitted.

```
location interface ATMW-LI
location type ATM
operations
    default(),cash():money
    acco():ACCOUNT
    give(n:money) post cash() = old cash()-n
events     withdraw(n:money)
end interface
```

The event that is being required is self-evident and, as we shall see in the next section, refers to the business activity for which we have already defined coordination laws. When this interface is instantiated, this event can be refined in many different ways depending on the actual machine at which the business activity is being performed: the pressing of a button in the keyboard, the filling of a menu on the screen, etc. The parameter of the event will also need to be provided on instantiation.

The ATM is required to make available two services: the amount of cash available inside the machine and the default maximum amount that the machine gives if there is no connection to the account. The ATM service is also required to make available the number of the account that is currently being serviced. This data will have been stored upon identification through the ATM card. We will see in the next section that location (and coordination) interfaces are instantiated in run-time to services that may be running, i.e. instantiation does not mean creation of a service. In the case at hand, the instance of ATM will be the service that will have been running when the ATM was "switched on" and that will have accepted and authenticated the card involved in the first activity of the specific banking process at stake.

The location interface that applies to the bank is as follows:

```
location interface rBANKW-LI
location type BANK
operations internal(n:money; a:ACCOUNT)
           maxatm(a:ACCOUNT): money
end interface
```

That is, the bank is required to be make available, for every account, the maximum amount that can be debited from an ATM, as well as accommodate executions of withdrawals internally. This is because we want to be able to move withdrawals to the bank when they are requested at the ATM and there is no communication between the two locations.

These two location interfaces are brought together in the location law that defines the distribution logic of the withdrawal activity when performed at an ATM:

```
location law ATMW-LL
locations  bank: rBANKW-LI; atm: ATMW-LI
rules
when  atm.withdraw(n) &
      BT(atm,bank)
      with n ≤ bank.maxatm(atm.acco()) &
           n ≤ atm.cash()
      do   atm.give(n)
when  atm.withdraw(n) & ¬BT(atm,bank)&
          REACH(atm,bank)
      let  N=min(atm.default(),n) in
      with N ≤ atm.cash()
      do   atm.give(N)
      mv   bank.internal(N, atm.acco()))
end law
```

As in coordination laws, location laws declare a number of partners (called locations) and their interfaces. The ECA rules that we use for describing the distribution logic in location laws differ from the ones used in coordination laws because the composition logic does not require the communication and reachability status to be taken into account. On the contrary, in location laws, we need to take into account the properties of the context in which the trigger occurs, the condition needs to be evaluated, and the action needs to be performed.

Indeed, as neither the presence nor the quality of communication can be taken for granted in location-aware business components, we have to take explicit account of the communication status between any involved interfaces using their locations. For instance, depending on whether given locations are in touch, either a full composition of operations is performed across all locations involved thus synchronising the services in execution at these locations, or just a composition of the operations available at the location where the trigger is perceived can be performed.

This dependency is made explicit through the use of BT. In the location law, two different rules are considered depending on whether the two locations are in touch when the request for the withdrawal is detected. Notice that the distinction is made at the level of the trigger (the event of the ECA), not the guard (condition). This is because each case needs to be treated differently, in particular through different guards: when BT holds, the guard concerns upholding the maximum withdrawal permitted by the bank at an ATM whereas, when BT does not hold, it is the maximum allowed by the ATM itself that needs to be upheld.

The fact that two locations are not "in touch" (BT) does not mean that one cannot be reached from the other (REACH). Reachability allows for mobility of services, namely for service execution to be moved to other locations as an instance of another service. In the case that concerns us, even in the absence of communication with the bank, ATMs can provide a limited amount of cash as long as there is a protocol with the bank for remote/delayed transmission of the corresponding withdrawal. The operations that continue the execution of the activity at a different location are declared under *mv* whereas those that are executed locally are identified under *do* as usual.

Notice that what is being moved for execution at the bank concerns a full withdrawal service, not the elementary debit operation that we discussed in the previous section. Indeed, the required service needs to be executed in the right context, which means taking into account the coordination and location rules that apply, internally at the bank, to that specific client and account. The way the service is moved from the ATM to the bank is left unspecified: it should be handled at the level of the definition of the location types, namely the topology of movement that applies. In the case of current Web services, these are rather trivial situations as reachability is, once again, handled at the level of network addresses. In our example, this movement can be just the storage of a request until communication becomes available (lightweight mobility), or the print out of instructions that are delivered in hand at the bank and executed on arrival at the end of the day (strong snail mobility), just to name a few and stress that we are modelling services that are not necessarily deployed over the Web!

As discussed in the next section, the transaction to be executed may involve whatever operations are required by the composition logic through the coordination rules that react to the same trigger. Indeed, the location rules above are not concerned with the contracts that the customer has with the bank with respect to withdrawals from the specific account that is involved as a partner, just as the coordination rules discussed in the previous section were not concerned with distribution. This separation of concerns is, precisely, what the paper aims to explain.

Before we discuss the integration of separately modelled concerns, consider a few more examples that illustrate other situations. For instance, consider the situation in which the request for the withdrawal is made at a branch of the bank, although not necessarily the one in which the account is held. We still need two location interfaces because two locations are involved:

```
location interface BRW-LI
location type BANK
operations
    cash():money
    give(n:money) post cash() = old cash()-n
events      withdraw(n:money; a:ACCOUNT)
end interface
```

```
location interface BANKW-LI
location type BANK
end interface
```

In this case, nothing is required of the bank location that concerns the distribution logic; only the coordination rules will apply as discussed in the next section. This becomes evident in the location law itself:

```
location law BRW-LL
locations  bank: BANKW-LI; branch: BRW-LI
rules
when  branch.withdraw(n,a) &
      BT(branch,bank)
      with n ≤ branch.cash()
      do   branch.give(n)
end law
```

In this case, there is no location rule for the situation in which the branch is not in touch with the "bank", i.e. with the location in which the account is held. This means that, in those circumstances, the request for the withdrawal is not recognised, i.e. does not constitute a trigger (the clerk at the branch just says "sorry: the system is down again"...)

Consider now a different business activity – identification. At an ATM, two locations are involved: the ATM itself and the card.

```
location interface ATMId-LI
location type ATM
operations
    acco():ACCOUNT;
    cust():CUSTOMER;
    accept(c:CARD) post acco()=ac(c) & cust()=ct(c)
events      enter(n:PIN)
end interface
```

```
location interface CARD-LI
location type CARD
operations attempts():nat
           code():PIN
           reject post attempts() = old attempts()+1
           accept post attempts()=0
end interface
```

The interface for the ATM detects the entering of a pin number. as an event. As elementary services, it involves the acceptance of a card, which implies retrieving from the card the identities of the account and the customer. This is done through operations *ac:CARD→ACCOUNT* and *ct:CARD→CUSTOMER* available at the level of the data types provided as part of the underlying business model. On the side of the card, elementary operations handle attempts at guessing the code that is stored.

The corresponding location law is pretty intuitive:

```
location law ATMId-LL
locations  atm: ATMId-LI; card: CARD-LI
rules
when  enter(n) &
      BT(atm,card)
      with card.attempts() ≤ 3
      do   if n = card.code()
           then card.accept() &
                atm.accept(card)
           else card.reject()
end law
```

Notice that, in this case, BT means that the ATM is able to recognise the card and, hence, "communicate" with it, namely to extract information from it as done through the action *accept*. If the card is not recognised, then the trigger is not recognised either and the evaluation of the guard is not even attempted.

5. INTEGRATION OF CONCERNS

So far we proposed a set of semantic primitives through which we can separate two different concerns in business modelling: the coordination mechanisms that should be put in place to compose services (composition logic or layer) and the location-aware aspects that handle the dependency on the business channels across which services are distributed (distribution logic or layer).

This separation of concerns seems to be rather intuitive. As a business activity, a withdrawal from a bank account should involve a number of partners that execute required services in a coordinated way, i.e. according to certain logic, regardless of where they are located. For instance, the use of a credit facility is part of a business contract between the customer and the bank regardless of the channel through which withdrawals are made. Likewise, the limitations that the absence of communication between an ATM and a bank imposes on the activity is independent of the existence of a credit allowance.

This is why it is important to support this separation of concerns at the level of business modelling. On the one hand, each dimension can be refined independently of the other. On the other hand, changes in one dimension can be done without interfering with decisions made in the other.

Being able to model these concerns separately does not mean that they are independent. The way a business activity is performed within a process system emerges from the coordination and location laws that jointly apply to that activity. In this section, we discuss this mechanism of emergence, i.e. we are concerned with the away both concerns get integrated in a model of the business activity as it ends up being executed.

As an example, consider the withdrawal once again. At run-time, the way the withdrawal is processed is determined not by independent partners and locations but by *located partners*: for instance, *cust@atm* and *acco@bank*. That is, both coordination and location interfaces need to be instantiated by the same run-time services. In particular, because the ATM component identifies a customer and an account, we have *cust=atm.cust()* and *acco=atm.acco()*, i.e. a single customer service and a single account service. This makes it clear that the business partner that is involved in the activity is not necessarily the person standing in front of the ATM but the customer identified in the card.

To be more precise, the instantiation of the coordination and location laws means binding the coordination and location interfaces to services that are running on the current system configuration. Hence, in the case of a withdrawal, we will have services running: one that binds *cust* and *atm*; the other binds *acco* and *bank*.

As already mentioned, these services are not necessarily disjoint or independent, and they are not necessarily created upon instantiation. For instance, as discussed in section 3, *acco* may be a service running autonomously within *bank*. On the other hand, the ATM service *atm* will have started when the ATM was switched on; when the binding of the location interface *ATMW-LI* takes place, it will have a context in which *atm.acco()* and *atm.cust()* will hold the identities of the account and customer to which the withdrawal applies. This is because, through the location law *ATMId-LL*, this data will have been retrieved from the card during the identification activity. Moreover, the binding also establishes that the value of *cust.owns(acco)* is *true*. Notice that, at a branch, the binding of *cust* would not necessarily establish this equality: in the case of the ATM, it is the use of the card that authenticates the pair *(cust.acco)*. This is another reason in support of making business processes location-aware.

The way a process activity like a withdrawal interacts with these services in described in the coordination and location rules according to the events that are detected in the run-time configuration. For instance, the event that triggers the withdrawal business activity instantiates as *atm.withdraw(n)* in the location interface and *cust.withdraw(n,acco)* in the coordination interface Assuming that the coordination law that is active in the run-time configuration is *SW-CL* (see section 3), the occurrence of the event is subject to the following rules:

```
when  cust.withdraw(n,acco)
      with acco.balance() ≥ n &
           cust.owns(acco)
      do   acco.debit(n)

when  atm.withdraw(n) & BT(atm,bank)
      with n ≤ bank.maxatm(atm.acco()) &
           n ≤ atm.cash()
      do   atm.give(n)

when  atm.withdraw(n) & ¬BT(atm,bank)& REACH(atm,bank)
      let  N=min(atm.default(),n) in
      with N ≤ atm.cash()
      do   atm.give(N)
      mv   bank.internal(N,atm.acco()))
```

The joint execution of ECA rules that we have in mind, as formalised in [14], takes the conjunction of the guards and the parallel composition of the actions (i.e. the union of the corresponding synchronisation sets) when BT holds. When the located partners are not in touch, i.e. cannot communicate, the coordination rules do not apply. As a result, the rules according to which a withdrawal is performed are:

```
when  atm.withdraw(n) & BT(atm,bank)
      with n ≤ acco.balance() &
           n ≤ bank.maxatm(acco) &
           n ≤ atm.cash()
      do   atm.give(n)  &
           acco.debit(n)

when  atm.withdraw(n) & ¬BT(atm,bank)& REACH(atm,bank)
      let  N=min(atm.default(),n) in
      with N ≤ atm.cash()
      do   atm.give(N)
      mv   bank.internal(N,acco)
```

That is, when the ATM is in communication with the bank, the withdrawal is performed according to the coordination rule of a standard withdrawal and the location rule of the ATM. Notice, however, that *cust.owns(acco)* holds as a result of the binding and, hence, was omitted from the "with" condition. The need for communication is obvious in the guard condition, which requires the balance of the account to be checked and the action, which requires the account to be debited. In the case of the joint execution of the guard, BT is necessary to ensure synchronous, atomic execution of the reaction. Notice that synchronous execution does not involve REACH because the service is not being moved from one location to another: both services are executed, each in its location, but atomically, which is what requires communication. Naturally, this semantics requires a proper distributed transaction management system to be in place. See [20] for transaction protocols in the scope of Web services.

Summarising, as claimed in section 2, our approach is activity-oriented in the sense that, for each activity within a business process, we identify which are the location and coordination concerns that apply to the business entities involved, and how they are put together to enforce the business process logic (e.g. the activity ordering). In general, there is a 0-N correspondence between each business process activity and coordination / location laws. That is, depending on the semantics of each activity, we may have no coordination laws (which is the case of identification in the example) or one or more coordination laws (case of withdrawals); and the same for location laws.

We have to emphasize that, depending on the business entities involved in a specific activity, not every law applies at each configuration. Determining which laws should apply and, for those that apply, how the business entities instantiate the interfaces (location and coordination), and how the corresponding instantiated coordination and location laws bind the entities together with contracts, is out of the scope of this paper. See [5,6] for configuration management primitives that apply to coordination laws. In what concerns location laws, we are now developing similar configuration primitives.

6. CONCLUDING REMARKS

In this paper, we discussed a service-oriented architectural-based approach that addresses current challenges in modern business process modelling for reflecting dynamic cross- and intra-organisational interactions as well as dependencies on the business channels and networks over which organisations operate. Our approach is inspired in the rich set of specifications that is currently available for software development over Web services, i.e. "software that can process XML documents it receives through some combination of transport and application protocols" [31]. Languages and techniques as made available by BPEL4WS [9], WS-Coordination [35] and WS-Transaction [36], *inter alia*, remain too close on this narrow view of services that need to be located and invoked over the Web using addresses and referencing mechanisms that identify where services can be found using a given protocol like TCP or HTTP. As a consequence, they offer little support to the higher-levels of abstraction in which business rules and organisational infrastructures need to be modelled.

This is why we decided to distance ourselves from both the XML-centred view of information exchange, and the Web-oriented notions of location and reference protocols. Our proposal addresses a rule-based approach to business modelling and addresses a space in which locations correspond to business entities and channels organised according to a given organisational communication and distribution network.

The semantic primitives that we proposed for business modelling capture structural features of architectural connectors in separating concerns and addressing business rules as first-class entities. Following our approach, the aspects that relate to the way business rules determine how the services involved in a business activity need to be orchestrated fall under what we call "coordination laws". These are semantic primitives that are used for modelling the "service composition layer" of service-oriented architectures or, for short, their "composition logic".

In what concerns the "distribution logic" that captures the dependency on the business channels and networks (e.g. properties of the computational platform and communication network, mobility of devices/sensors, inter alia), we proposed a similar approach based on explicit connectors we called location laws. As with coordination laws, these connectors can be superposed dynamically and evolved independently of the other business aspects, allowing systems to self-adapt or be adapted to changes that occur at the distribution level without interfering with the core business policies.

The semantics of both the composition and distribution logic, and of coordination and location laws, builds on recent work around CommUnity, a formal approach that we have been developing for architectural description [14]. CommUnity includes primitives that capture distribution and mobility aspects [19], and explicitly separate between components computation, coordination and distribution/mobility. Besides recently forwarded operational semantics—including graph transformations, Tile and rewriting logic—the main strength of CommUnity lies in its logic of interactions, which is based on Category Theory [13]. CommUnity is also endowed with a software tool for editing, simulating and validating distributed software architectures. Extensions of CommUnity towards context-aware computing are now being explored that will further enrich this architectural approach.

We are currently working on more case studies in order to consolidate and validate this service-oriented architectural approach. We are also collaborating with ATX Software, the IT company with whom we developed the Coordination primitives, on the methodological aspects of location laws; one of our main goals is to develop a deeper understanding and classification of business rules so that semi-automatic derivation of coordination and location laws can be ultimately achieved. In this sense, the work forwarded in [26] on classifying Web Services-oriented rules could be a significant input for us. Last but not least, extensions to modelling languages like the UML with coordination and distribution laws are also being investigated at Leicester.

Acknowledgements

N..Aoumeur was supported by the European Commission through the contract IST-2001-32747 (AGILE: Architectures for Mobility). C.Oliveira was supported by Fundação para a Ciência e Tecnologia, Portugal, through the PhD Scholarship SFRH/BD/6241/2001, and the European Science Foundation through the Scientific Network RELEASE. The authors would like to thank P.Kosiuczenko and A.Lopes for many insights and suggestions on the work reported in this paper.

7. REFERENCES

1. W.Aalst, A.T.Hofstede and M.Weske, "Business Process Management: A Survey", in *International Conference on Business Process Management (BPM 2003),* LNCS 2678, Springer 2003, 1-12.
2. L.Abom, "Frameworking RM-ODP in Banking", in A.M.Cordeiro and H.Kilov (eds) *WOODPECKER 2001*, ICEIS Press 2001.
3. R.Allen and D.Garlan, "A Formal Basis for Architectural Connectors", *ACM TOSEM*, 6(3), 1997, 213-249.
4. L.Andrade, J.L.Fiadeiro, A.Lopes and M.Wermelinger, "Coordination for Distributed Business Systems", in *Information Systems for a Connected Society*, J.Eder, R.Mittermeir and B.Pernici (eds), University of Maribor Press 2003, 27-37.
5. L.F.Andrade and J.L.Fiadeiro, "Service-Oriented Business and System Specification: Beyond Object-orientation", in H.Kilov and K.Baclwaski (eds), *Practical Foundations of Business and System Specifications*, Kluwer Academic Publishers 2003, 1-23.
6. L.F.Andrade and J.L.Fiadeiro, "Composition Contracts for Service Interaction", *Journal of Universal Computer Science*, in print.
7. A Baina, S. Tata, and K. Benali, "A Model for Process Service Interaction", in *International Conference on Business Process Management (BPM 2003),* LNCS 2678, Springer 2003, 261-275.
8. BizTalk Orchestration – a new technology for orchestrating business interactions, Microsoft Research 2000.
9. Business Process Execution Language for Web Services, version 1.1, May 2003, IBM
10. L.Cardelli and A.Gordon, "Mobile Ambients", in Nivat (ed), *FoSSACs'98*, LNCS 1378, 140-155, Springer-, 1998.
11. F.Curbera, R.Khalaf, N.Mukhi, S.Tai and S.Weerewarana, "The Next Step in Web Services", in [27], 41-47.
12. T.Elrad, R.Filman and A.Bader (Guest editors). Special Issue on Aspect Oriented Programming. *Communications of the ACM* 44(10) 2001.
13. J.L.Fiadeiro, *Categories for Software Engineering*, Springer 2004.
14. J.L.Fiadeiro, A.Lopes and M.Wermelinger, "A Mathematical Semantics for Architectural Connectors", in *Generic Programming*, R.Backhouse and J.Gibbons (eds), LNCS 2793, Springer 2003, 190-234.
15. P.Kardasis and P.Loucopoulos, "Expressing and Organising Business Rules", *Information and Software Technology*, in press.
16. S.Katz, "A Superimposition Control Construct for Distributed Systems", *ACM TOPLAS* 15(2), 1993, 337-356.
17. Z.Kleppe, J.Warmer and W.Bast, *MDA Explained: The Model Driven Architecture--Practice and Promise*, Addison-Wesley 2003.
18. A.Lindsay, D.Downs and K.Dunn, "Business Processes – attempts to find a definition", *Information and Software Technology* 45(1):1015-1019, 2003.
19. A.Lopes and J.L.Fiadeiro, "On how Distribution and Mobility interfere with Coordination", in *Recent Trends in Alge-*

braic Development Techniques, M.Wirsing, D.Pattinson, R.Hennicker (eds), LNCS 2755, Springer 2003, 343-358.

20. M.Little, “Transactions and Web Services”, in [27], 49-54.

21. P.Loucopoulos, “The S3 (Strategy-Service-Support) Framework for Business Process Modelling”, in *CAiSE Workshops – Information Systems for a Connected Society*, CEUR Workshop Proceedings vol. 75, Technical University of Aachen (RWTH), 2003.

22. J.Magee and J.Kramer, "Dynamic Structure in Software Architectures", in *4th Symp. on Foundations of Software Engineering*, ACM Press 1996, 3-14.

23. A.Maurino, B.Pernici and F.Schreiber, “Adaptive Channel Behavior in Financial Information Systems”, in *CAiSE Workshops – Information Systems for a Connected Society*, CEUR Workshop Proceedings vol 75, Technical University of Aachen (RWTH), 2003.

24. G.Meredith and S.Bjorg, “Contracts and Types”, in [27], 41-47.

25. B.Orrinsi, J.Yang, and M.Papazoglou, “A Framework for Business Rule Driven Web Service Composition”, in *Proc. of Conceptual Modeling for Novel Application Domains*, LNCS 2814 Springer 2003, 52-64.

26. B.Orrinsi, J.Yang, and M.Papazoglou, “A Framework for Business Rule Driven Web Service Composition”, in *Proc. of Conceptual Modeling for Novel Application Domains*, LNCS 2814 Springer 2003, 52-64.

27. M.Papazoglou and D.Georgakopoulos (guest editors), Special Issue on Service-Oriented Computing, *Communications of the ACM* 46(10), 2003.

28. G.-C.Roman, C.Julien and J.Payton, “A Formal Treatment of Context-Awareness”, *Proc. FASE 2004*, LNCS 2984, 12-36, Springer-Verlag, 2004

29. D.Rosca and C.Wild, “Towards a Flexible Deployment of Business Rules”, *Expert Systems with Applications* 23:385--394, 2002.

30. M.Shaw, "Procedure Calls are the Assembly Language of Software Interconnection: Connectors Deserve First-Class Status", in D.A. Lamb (Ed.), *Studies of Software Design,* LNCS 1078, Springer 1996.

31. W.Vogel, “Web Services Are Not Distributed Objects”, *IEEE Internet Computing* 2003.

32. J.Yang, “Web Service Componentization”, in [27], 35-40.

33. W.Wan-Kadir and P.Loucopoulos, “Relating Evolving Business Rules to Software Design”, *Journal of Systems Architecture*, 2003.

34. *Web Services architecture overview – the next stage of evolution for e-business,* September 2000, http://www.ibm.com/developerworks/web/library/w-ovr/

35. *Web Services Coordination, version 1.0,* http://www.ibm.com/developerworks/web/library/ws-coor/

36. *Web Services Transaction, version 1.0,* http://www.ibm.com/developerworks/web/library/ws-transpec/

A Lightweight Approach for QoS–Aware Service Composition

Gerardo Canfora,
Massimiliano Di Penta
RCOST - Research Centre on Software Technology
University of Sannio,
Department of Engineering
Palazzo ex Poste, Via Traiano
82100 Benevento, Italy
canfora@unisannio.it,
dipenta@unisannio.it

Raffaele Esposito,
Maria Luisa Villani
RCOST - Research Centre on Software Technology
University of Sannio,
Department of Engineering
Palazzo ex Poste, Via Traiano
82100 Benevento, Italy
r.esposito@unisannio.it,
villani@unisannio.it

ABSTRACT

One of the most challenging issues of service–centric software engineering is the QoS–aware composition of services. The aim is to search for the optimal set of services that, composed to create a new service, result in the best QoS, under the user or service designer constraints. During service execution, re-planning such a composition may be needed whenever deviations from the QoS estimates occur. Both QoS–aware composition and re-planning may need to be performed in a short time, especially for interactive or real–time systems. This paper proposes a lightweight approach for QoS–aware service composition that uses genetic algorithms for the optimal QoS estimation. Also, the paper presents an algorithm for early triggering service re-planning. If required re-planning is triggered as soon as possible during service execution. The performances of our approach are evaluated by means of numerical simulation.

Categories and Subject Descriptors

K.6.3 [**Management Of Computing And Information Systems**]: Software process; G.1.6 [**Numerical Analysis**]: Constrained optimization; H.3.5 [**Information Storage And Retrieval**]: Web-based services

General Terms

Algorithms, Measurement

Keywords

Quality of Service, Web Service Composition, Genetic Algorithms

1. INTRODUCTION

Web services constitute a promising technology landscape for software engineering. During the last 20 years, component based software engineering aimed at the application of principles used for years in other engineering disciplines to software development activities. In fact, electronics, mechanics and other engineering systems are commonly built by assembling pre-defined components, such as memories, CPUs, etc.

In the same way, the spread of reusable software components avoids software engineers to "re-invent the wheel" each time. Building software systems by "gluing" components enables them to concentrate more on the problems their systems aims to solve. This scenario changes when moving from component-based systems to service-centric systems, where a functionality is realized by searching, composing and executing services. In the particular case of web services, this is done by using a set of XML-based standards, known as UDDI, WSDL and SOAP [16].

The service-centric systems scenario poses several additional challenges with respect to component-based software engineering. First and foremost, in a component-based software system components are physically integrated and, except for distributed systems, they are executed as a whole in the end-user's environment. This is not usually the case of web services as they are executed on the service provider server, thus raising issues on the run-time service (and of course network) availability and performances.

Secondly, several services may be available with the same function (we call them *semantically equivalent* services), however they surely exhibit different Quality of Service (QoS). According to Std. ISO 8402 [9] and ITU [10], QoS may be defined in terms of attributes such as price, response time, availability, reputation (further details can be found in Cardoso's PhD thesis [2]). Moreover, it may be possible to have some domain-specific QoS attributes (e.g., a temperature service could have QoS attributes such as precision or refresh frequency). The choice between different but semantically equivalent services is a function of such QoS attributes: one may decide to choose the cheapest service, the fastest, or maybe a compromise between the two. Moreover, an user may specify constraints on the values of some attributes (e.g., the price cannot be higher than a given value), which could influence the choice. On the other hand, the service provider can estimate ranges for the QoS attribute values as part of the contract with potential users. Also, the QoS guarantees for the same service could be customer-dependent, and so they would apply each to a different instance of that service. For example, an user that buys a service at a given price is not expected to get a response time below a given threshold.

Once service annotation and matching mechanisms are available, a semantic description of the service may be included in the user system as a reference to it. Such a description, that we call an *abstract service*, corresponds to a specific functionality, while not necessarily to a single service implementation. At run-time, using a matching algorithm, this description is used to retrieve some services (that we will refer to as *concrete services*), and then select one among them that meets the constraints and maximizes our QoS objective function. Several matching approaches have been proposed in literature, see for example Paolucci et al. [13].

A composite service is a service resulting from a composition of other services whose interaction is described by some workflow description language (e.g., BPEL4WS [1]). The component services can be, on their own, *abstract services*, therefore two different problems need to be solved:

- determine the QoS of a composite service as a function of the QoS of its components; and
- determine the set of *concrete services* that maximize the QoS of the composite service. In other words, for each *abstract service* of the workflow, determine a concretization such that the total QoS is maximized and the global constraints are met.

Clearly, the QoS values of the single service components used for computation may be estimations in turn, declared by each service provider or obtained by computing statistics during previous executions of the service. At run-time, the actual (measured) QoS values may deviate from the estimate ones or, simply, one of the services may not be available. Thus the composite service may have to be re-planned, so to still meet the constraints and maximize the QoS. Some approaches have been proposed in literature [17] to this aim.

All of this, from the QoS-aware composition to re-planning, often needs to be performed very quickly. Especially for interactive systems, long delays may be unacceptable. For example, the user of a booking ticket system might not want to wait for a long time while the system searches for candidate services offering flight tickets with the lowest booking fare. Gaining a few cents after several minutes of waiting may make the user disappointed. Even for some non-interactive service a fast composition just before execution may be desired: performing service composition long before execution may lead to unattended results (e.g., some services may not be available anymore or, conversely, new, more convenient services can be available).

The paper aims are the following:

1. it proposes an approach for a quick, coarse–grained QoS-aware service composition, where some of the composition rules are the same as those proposed by Cardoso [2];
2. when the QoS optimality is more relevant than the performance of the service composition algorithm, like for scientific computations, an alternative approach is proposed;
3. finally, the paper proposes a re-planning algorithm aiming to anticipate the re-planning decisions as soon as possible during the composite service execution. Re-planning is then performed on a slice of the original workflow, avoiding to unfold loops that could worsen the performances.

Numerical simulations have been used to evaluate the results.

The remainder of this paper is organized as follows. After a review of the literature in Section 2, Section 3 details the approach we propose. In particular, the QoS composition rules are shown, the evolutionary approach for QoS-aware composition is described and, finally, the re-planning algorithm is detailed. Section 4 describes the toolkit aiming to support the work, while Section 5 reports and discusses results obtained in the simulations. Finally, Section 6 concludes.

2. RELATED WORK

QoS-aware discovery and composition of services has been recognized as a crucial aspect in the web services era, where companies are starting to deliver their products as services over the Internet and the service-oriented architecture paradigm has become a new reference for the software business. In this context, providers need ways to express their quality guarantees on the service being advertised, and technological support should be given to customers to search for and select the best available service. Furthermore, satisfaction of the quality requirements he/she specified should be assured during execution. This is more difficult for composite services, where the overall QoS relies on that of each component service, and it depends on how services are integrated and interact with each other. General issues for Web Services QoS are discussed in a paper by Ludwig [11].
Some formalism for the service QoS specification and Service Level Agreements (SLAs) has been provided, such as the IBM's Web Service Level Agreement (WSLA) language [12], or the Web Services Offer Language (WSOL) [15]. On the other hand, some web service orchestration languages are being proposed as standard, like Business Process Execution Language for web services (BPEL-4WS) [1]. Nevertheless, the currently available workflow technology still lacks of facilities for a complete QoS estimation, management and monitoring for processes. In fact, most established solutions in the area of workflow focus only on time management or load balancing and web-service-based systems that account for other QoS criteria are still being experimented ([3], [14], [17]). In particular, some on-going research activities deal with QoS prediction and dynamic adaptation of the workflow to face unexpected QoS progress during execution. In this respect, our work is positioned within that of Cardoso [2], and that of Zeng et al. [17].
The former proposes a mathematical model for workflow QoS computation, described by some metrics aggregation functions which are defined for time, cost, reliability, and fidelity. The meaning of each metrics is precisely given. The model uses stochastic information indicating the probability of transitions being fired at run-time, which can be initially set by the designer, wherever possible, and then periodically adjusted, based on data on previous executions stored in the workflow system log. The same method is also used to re-compute tasks' QoS. The QoS computation algorithm (SWR) proposed by Cardoso consists of applying a set of reduction rules to the workflow until one atomic task is obtained. We also use a reduction approach for our workflow QoS estimation, because of the advantage of fast computation, and mostly consider the same aggregation functions for each metrics, some of which are reported in Table 1. However, Cardoso's method does not consider optimal binding of the service components nor re-negotiation at run-time.

In particular, the loop reduction rule does not seem to be suitable for re-planning. In fact, this rule is based on a probability for the feedback transition to fire, thus it does not allow distinguishing the workflow part to be re-planned during execution, in case of errors happening within the loop. We propose a different reduction rule for the loop construct which enables dynamic service binding and re-negotiation.
The work of Zeng et al. focuses on dynamic and quality-driven selection of the service implementations of the workflow that accounts for local and global quality constraints and user preferences. Most notably, they propose a global planning approach to reach overall QoS optimality through linear programming techniques. The method is discussed, and empirical data given, under the assumptions of the workflow being acyclic, the constraints and objective function being linear, and the workflow consisting of one execution path, for which the QoS estimate is made. Then the method could be generalized by unfolding loops, based on the estimated number of iterations, binding a concrete service to each task by considering the "hot path" for that task, i.e. the most frequently executed path containing that task, while the constraint on linearity is overcome by applying logarithmic transformation functions. While this approach is quite effective with respect to reaching QoS optimality, in case of a complex workflow with branches and frequent loop iterations the concretization process they propose seems to be less efficient. In this paper, we discuss the trade-off between efficiency of the concretization algoritm and optimality of the solution obtained. Also, we extend their method for workflow re-planning.
Web-Flow [14] and eFlow [5] are workflow management systems that offer some support to selection of services according to quality constraints. However, these constraints are only applied at task level. In particular, eFlow allows to bind a service implementation to a *generic node* at run-time through a *search recipe*, while Web-Flow includes an exception handling mechanism, based on the Event-Condition-Action paradigm, triggered by the violation of constraints or other events like service faults, that may occur during process execution. A model for exception analysis, prediction and prevention in business processes, based on data warehousing and mining techniques, was presented by Casati et al. [4]. The exceptional events are stated by the user and are defined by conditions over process execution data. No dynamic recovery is addressed.
Finally, solutions for resource allocation and performance management that can be borrowed from the distributed systems area have been suggested [11], along with an analysis on web services-specific issues.

3. APPROACH DESCRIPTION

As stated in the introduction, the proposed approach is mainly devoted to allow for a fast, overall computation of the QoS of a composite service, as well as to determine the optimal set of *concrete services* to be bound to the *abstract services* composing the workflow. The model also enables re-planning.

In the sequel we shall consider a composite service S of n *abstract services*, $S \equiv \{s_1, s_2, \ldots, s_n\}$, whose structure is defined through some workflow description language. Each component s_i can be bound to one of the m *concrete services* $cs_{i,1}, \ldots, cs_{i,m}$, which are functionally equivalent.

3.1 Computing the QoS of Composite Services

This section describes our approach, hereby referred as the *unlooping* approach, for computing the QoS of composite service. Similarly to what proposed by Cardoso [2], for a Switch construct in the workflow, each `case` statement is annotated with the probability to be chosen (see Figure 1-a). For example, for a workflow containing a Switch composed of two Cases, with costs C_1 and C_2 respectively and probabilities p and $1-p$, the overall cost is computed as follows:

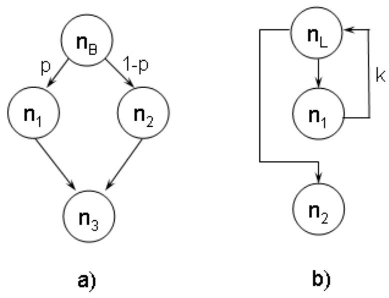


Figure 1: Workflow annotation: a) Switch b) Loop

$$p\,C_1 \;+\; (1-p)\,C_2 \qquad (1)$$

Clearly, probabilities are initialized by the workflow designer, and then eventually updated considering the information obtained by monitoring the workflow executions.
Loops are handled differently from Cardoso [2], that basically proposes to adopt a mechanism (based on the probabilities of entering/exiting the Loop) as for the `switch` construct. Our approach is more similar to what proposed by Zeng et al [17], i.e., Loops are annotated with an estimated number of iterations k. Instead of unfolding Loops (like Zeng et al.), here the QoS of the Loop is computed taking into account the factor k (see Figure 1-b) . For example, if the Loop compound has a cost C_l, then the estimated cost of the Loop will be $k\ C_l$.

This approach for handling Loops presents two advantages:

- It allows for a quick computation of the overall workflow QoS, without the need to unfold Loops;
- The estimated QoS accounts for the estimated number of Loop iterations.

Given a *concretization* of a composite service, i.e., a composite service description where each *abstract service* has been bound to one of its corresponding *concrete services*, the overall QoS can be computed by applying the rules described in Table 1, which shows an aggregation function for each pair workflow construct and QoS attribute. While for some standard QoS attributes the aggregation function has been explicitly specified ([17], [2]) there may be other attributes (for example, domain-dependent attributes) for which the aggregation function is user–specified (see the last row of Table 1).

It should be noted that the table is not complete (it only contains rules to be used in our examples) and, except that for Loops, the aggregation functions correspond to those proposed by Cardoso [2]. These functions are recursively defined on compound nodes of the workflow, although the table only shows their definition on sets of elementary tasks. Namely, for a Sequence construct of tasks $\{t_1, \ldots, t_m\}$, the *Time* and *Cost* functions are additive while *Availability* and *Reliability* are multiplicative. The Switch construct of

QoS Attr.	Sequence	Switch	Fork	Loop
Time (T)	$\sum_{i=1}^{m} T(t_i)$	$\sum_{i=1}^{n} p_{ai} * T(t_i)$	$Max\{T(t_i)_{i\in\{1\ldots p\}}\}$	$k * T(t)$
Cost (C)	$\sum_{i=1}^{m} C(t_i)$	$\sum_{i=1}^{n} p_{ai} * C(t_i)$	$\sum_{i=1}^{p} C(t_i)$	$k * C(t)$
Availability (A)	$\prod_{i=1}^{m} A(t_i)$	$\sum_{i=1}^{n} p_{ai} * A(t_i)$	$\prod_{i=1}^{p} A(t_i)$	$A(t)^k$
Reliability (R)	$\prod_{i=1}^{m} R(t_i)$	$\sum_{i=1}^{n} p_{ai} * R(t_i)$	$\prod_{i=1}^{p} R(t_i)$	$R(t)^k$
Custom Attr. (F)	$f_S(F(t_i)_{i\in\{1\ldots m\}})$	$f_B((p_{ai}, F(t_i))_{i\in\{1\ldots n\}})$	$f_F(F(t_i)_{i\in\{1\ldots p\}})$	$f_L(k, F(t))$

Table 1: Aggregation functions per workflow construct and QoS attribute

Cases $1, \ldots, n$, with probabilities $p_{a1}, \ldots, p_{an}$ such that $\sum_{i=1}^{n} p_{ai} = 1$, and tasks $\{t_1, \ldots, t_n\}$ respectively, is always evaluated as a sum of the attribute value of each task, times the probability of the Case to which it belongs. The aggregation functions for the Fork construct, are essentially the same as those for the Sequence construct, except for the *Time* attribute where this is the maximum time of the parallel tasks $\{t_1, \ldots, t_p\}$. Finally, a Loop construct with k iterations of task t is equivalent to a Sequence construct of k copies of t.

3.2 Searching for a solution with Genetic Algorithms

Determining the best concretization of a composite service is an optimization problem:

1. Maximize a fitness function of the available QoS attributes; and
2. Meet the constraints specified for some of the attributes. In particular, these are the global constraints, i.e. assertions on the overall QoS attribute values. Local constraints, i.e. constraints on each service s_i composing our service, need to be checked when choosing the set of candidate *concrete services* for s_i.

Finding a solution for the above problem is NP-hard [7]. In this case, different strategies can be adopted, for example integer programming [17] or Genetic Algorithms (GA). In our work we chose to adopt GA because the problem, as stated in our case, is well suited to be encoded with a genome and solved using GA evolution. Differently to linear programming approaches, GA does not impose constraints on the linearity of the QoS composition operators (and thus of objective function and constraints). This permits to adopt our approach for all possible (even customized) QoS attributes, without the need for linearization.

GA originated with an idea, born over 30 years ago, of applying the biological principle of evolution to artificial systems. Roughly speaking, a GA is an iterative procedure that searches for the best solution of a given problem among a constant-size population, represented by a finite string of symbols, named the *genome*. The search is made starting from an initial population of individuals, often randomly generated. At each evolutionary step, individuals are evaluated using a *fitness function*. High–fitness individuals will have the highest probability to reproduce.

The evolution (i.e., the generation of a new population) is made by means of two operators: the *crossover operator* and the *mutation operator*. The crossover operator takes two individuals (the *parents*) of the old generation and exchanges parts of their genomes, producing one or more new individuals (the *offspring*). The mutation operator has been introduced to prevent convergence to local optima, in that it randomly modifies an individual's genome (e.g., by flipping some of its bits, if the genome is represented by a bit string). Crossover and mutation are performed on each individual of the population with probability *pcross* and *pmut* respectively, where $pmut \ll pcross$. Further details on GA can be found, for example, in the Goldberg's book [8].

To let the GA search for a solution of our problem, we first need to encode the problem with a suitable genome. In our case, the genome is represented by an integer array with a number of items equals to the number of distinct *abstract services* composing our service. Each item, in turn, contains an index to the array of the *concrete services* matching that *abstract service*. Figure 2 gives a better idea of how the genome is made.

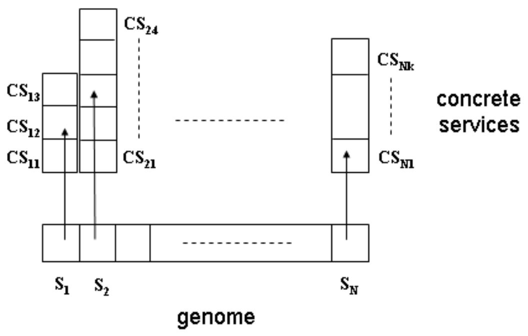


Figure 2: Genome Encoding

The crossover operator is the standard two-points crossover [8], while the mutation operator randomly selects an *abstract service* (i.e., a position in the genome) and randomly replaces the corresponding *concrete service* with another one among those available. Clearly, *abstract services* for which only one *concrete service* is available are taken out from the GA evolution.

The problem can now be modeled by means of a fitness function and, eventually, some constraints. The fitness function needs to maximize some QoS attributes (e.g., reliability), while minimizing others (e.g., cost). When user–defined, domain–specific QoS attributes are used, the specification of the fitness function is left to the workflow designer. For standard QoS attributes, we define a fitness function for a genome g as follows:

$$F(g) = \frac{w_1\ Availability\ +\ w_2\ Reliability}{w_3\ Cost\ +\ w_4\ Response\ Time} \tag{2}$$

where $w_1, \ldots, w_4$ are real, positive weighting factors. As shown, our fitness function is multi-objective. Different approaches have been proposed in literature to deal with this kind of fitness function [6]. Generally, calibrating weights is guided by observing the fitness function's landscape, as well as from the analysis of the evolution of the different factors.

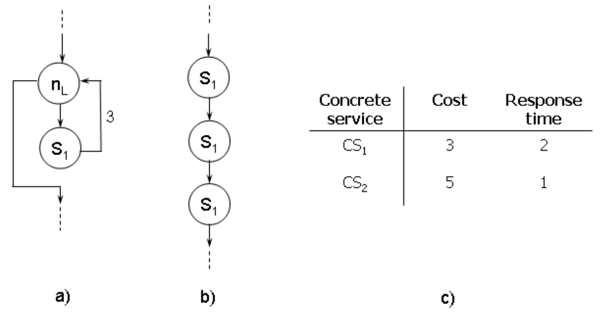
As mentioned above, constraints are assertions on the overall values of QoS attributes, e.g.:

$$\begin{aligned} Cost\ &<\ 50 \\ Time\ &<\ 100 \\ \frac{1}{2}\ Availability\ +\ \frac{1}{2} Reliability\ &>\ 0.95 \end{aligned}$$

When starting and evolving the GA, constraints need to be met anyway. This requires the genome initialization, the crossover operator and the mutator operator be modified. In fact, any randomly–generated genome that constitutes an individual of the starting population must be rejected (and thus generated again) if its QoS violates the constraints. Similarly, crossover and mutation operations must be roll-backed when, respectively, the offspring or the mutated individual violate the constraints.

3.3 Unfolding Approach: when better QoS is more essential than Quickness

The described approach permits to quickly find a solution to the QoS optimization problem. However, when QoS attributes are constrained, there could be better solutions at the price of a more expensive search.



Concrete service	Cost	Response time
CS_1	3	2
CS_2	5	1

Figure 3: When unfolding may be convenient

Let us consider, for example, a simple workflow shown in Figure 3-a), where the Loop is estimated to be executed 3 times. Suppose now that our optimization problem is given by:

$$\begin{aligned} F(x) = 1/(Cost + Time) \\ Time \leq 5 \end{aligned}$$

The GA described in Section 3.2 suggests that the *concrete service* CS_2 binds the *abstract service* S_1; the latter does not maximize the fitness function (that is equal to 1/18, while it would be 1/15 if CS_1 is chosen), although the constraints are met.

Let us suppose now that we unfold the Loops in our workflow, obtaining a workflow depicted in Figure 3-b. Let us consider a genome composed of all the nodes of the unfolded workflow, as represented in Figure 3-c. This encoding permits to have different bindings for different invocations of a service component, that can be useful to obtain a better QoS while meeting the constraints. In our example, a binding $\{CS_1, CS_2, CS_1\}$ will ensure the constraint to be met, and also produces a fitness value of $1/16 > 1/18$.

Clearly, as it will be shown in Section 5.1, Loop unfolding could mean an explosion of the genome size, and consequently of the time the GA requires to converge. Therefore, a tradeoff should be pursued. Interactive applications may accept a weaker QoS in favor of a short service negotiation time. Besides, for long–time–run scientific computations a long service negotiation time can be acceptable if, for example, we are composing a workflow that implements an algorithm to be run over weeks of computation time. In that case, the choice of the best combination of services, although requiring a longer time (e.g., some hours), could save days of computation time.

3.4 Triggering Service Re-planning

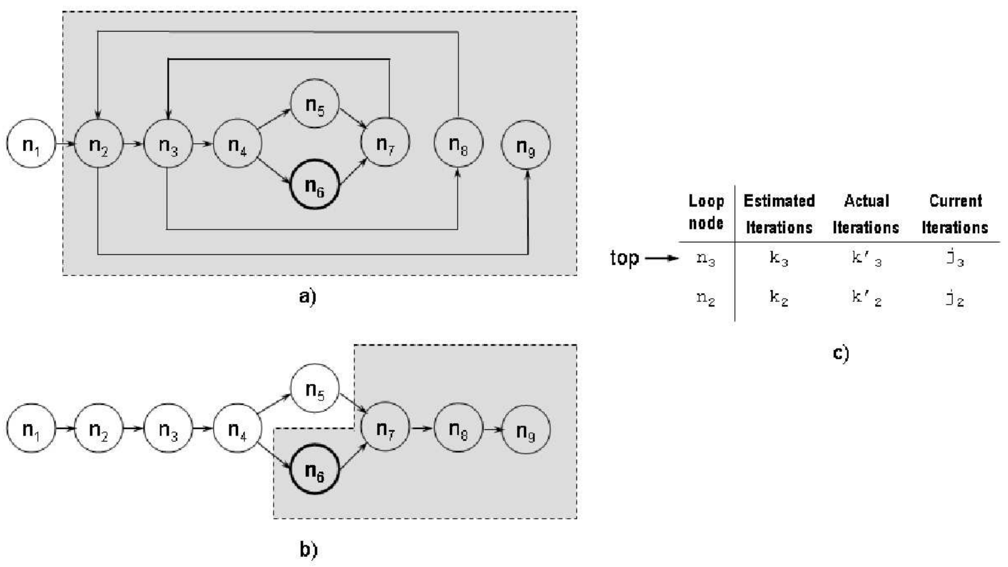
During workflow execution, the actual QoS may deviate from the estimated one, according to formulae shown in Table 1. Furthermore, there could be services not more available when invoked. In the two situations above, the slice of the workflow still to be executed may need to be re-planned.

The algorithm presented in Figure 5 describes the proposed re-planning triggering approach. Actually, this algorithm needs to be integrated with the workflow engine to allow measuring the actual QoS during execution and to perform re-planning when needed. The algorithm is described for any additive QoS attribute (e.g., cost) however it is still valid (with proper changes in the QoS formulae) for other (e.g., multiplicative) attributes. Also, for simplicity, the algorithm shows how re-planning works for constructs such as Loop, Switch, Sequence and invocation, while it can be easily extended to other constructs (e.g. Fork).

Given the overall estimated QoS (Q_{EST}), initially the actual workflow QoS (Q_{ACT}) is equal to it. Then, the workflow execution starts visiting the root node, and each node is recursively visited. Each time the absolute difference between the actual QoS and the estimated QoS is above the fixed threshold NTH, a re-planning is triggered.

For Loop nodes, the actual number of iterations k' is determined if possible (when the Loop exit is bound to a condition, this might not be possible), and the actual QoS is refined varying it by $(k' - k) * Q_{INNER}$ (i.e., considering that the number of iterations is varied by $k' - k$). In case this difference evaluates above the threshold, a re-planning is triggered. Then, the Loop inner node is visited k' times (or while the Loop condition is *true*), triggering re-planning each time this is necessary.

For Switch nodes, the actual Case to be executed (the $j-th$ one) is determined, and the Switch inner QoS (originally a weighted sum, as shown in equation (1)) is updated, considering, instead, only the



Loop node	Estimated Iterations	Actual Iterations	Current Iterations
n_3	k_3	k'_3	j_3
n_2	k_2	k'_2	j_2

Figure 4: Re-planning slice a) Node outside any Loop or Sequence b) Node inside a not yet estimated to be completed Loop c) Node inside completed Loops

QoS of the Case chosen.

For Sequence nodes, each child is visited, and re-planning is triggered each time the deviation of the actual QoS from the estimate is above the threshold. Finally, for service invocation nodes the measured QoS after service invocation is used to update the actual QoS.

3.5 Determining the re-planning slice

The function *triggerReplan()* invoked in the algorithm of Figure 5 acts on a slice of the original workflow, representing the set of nodes that still are to be executed.

Given a node n_p that, after its execution, triggers a re-planning, the slice to be re-planned is computed depending on the position of the node n_p in the workflow control structure.

- If the node is outside any Loop or Switch statement, all the nodes that follow n_p (including the node n_p itself) are part of the slice (Figure 4-a);
- If the node is part of a Switch, while outside any Loop, the slice will include again all the nodes that follow n_p, excluding the nodes of the alternative Cases (Figure 4-b);
- Whenever the workflow execution enters a Loop i, the following information is pushed on a stack: a reference to the Loop node, the estimated number of iterations k_i, the re-estimation k'_i (if available, otherwise $k'_i = k_i$), and the actual number of executed iterations j_i. When a re-planning is triggered inside a Loop, the slice is obtained starting from the most external, not yet *estimated to be completed*, Loop. A Loop i is estimated to be completed *iff* $j_i \geq k'_i$. Clearly, the resulting slice will contain, for such most external Loop, an estimated number of iterations equals to $k'_i - j_i$, i.e., only accounting for the iterations left to be executed.

Given the re-planning slice, the same approach described in Section 3.2 is used to find its (sub)-optimal concretization. However, this time the overall QoS that maximizes the fitness function while meeting the constraints is given by:

$$Q_{OVERALL} = Q_{TOT} + QoS(slice) \qquad (3)$$

i.e., the QoS of already executed nodes, plus the estimated QoS of the slice.

4. TOOL DESCRIPTION

We have implemented a tool prototype for a QoS-aware composition of services that we used for the experimentation of the presented approaches. Some of the components we realized could be integrated with a real web services orchestrator, while for the purpose of our experiments it was enough implementing a quite simple workflow engine simulator. The architecture of the tool is composed of:

- a *Service Repository*, where concrete services are stored along with their QoS information, and classified according to their semantic descriptions;
- a *Workflow Generation Tool*, used to automatically produce XML workflow representations of composite services. The workflow tasks are abstract services, while BPEL4WS-like constructs are used for the control flow specification. Also, it is possible to attach QoS constraints to workflow tasks;
- a *Concrete Workflow Builder*, for retrieving concrete services for the workflow tasks from the Service Repository, eventually according to local QoS constraints;
- a *Workflow QoS Estimator*, used for the overall QoS estimation of a concrete workflow;

```
$Q_{EST}$ ←Estimated overall QoS;
$Q_{ACT} \leftarrow Q_{EST}$;
$Q_{TOT} \leftarrow 0$ ;
$NTH$ ←replanning threshold;
$NODE \leftarrow$ Workflow root node;
visit(NODE);
begin function visit(node)
    switch node is type of do
        case loop
            $k \leftarrow$ Estimated loop iterations ;
            $k'$ Actual # of loop iterations ;
            $INNER \leftarrow$ Loop inner node
            $Q_{INNER} \leftarrow QoS(INNER)$;
            $Q_{LOOP} \leftarrow k' * Q_{INNER}$;
            $Q_{ACT} \leftarrow$
            $Q_{ACT} + (k' - k) * Q_{INNER}$ ;
            if $|Q_{ACT} - Q_{EST}| > NTH$ then
                replan($INNER$, $Q_{TOT}$) ;
            end
            for $j \leftarrow 1$ to $k'$ do
                visit($INNER$);
                if $|Q_{ACT} - Q_{EST}| > NTH$
                then
                    $Q_{TOT} \leftarrow Q_{TOT} -$
                    $ActQoS(INNER)$ ;
                    replan($INNER$, $Q_{TOT}$);
                end
            end
        case switch
            $j \leftarrow$ Case statement chosen ;
            $INNER \leftarrow$ Inner node of the
            $j - th$ case ;
            $Q_{SWITCH} \leftarrow QoS(node)$;
            $Q_{INNER} \leftarrow QoS(INNER)$;
            $Q_{ACT} \leftarrow$
            $Q_{ACT} - Q_{SWITCH} + Q_{INNER}$;
            visit($INNER$);
        case sequence
            foreach Node n in sequence do
                visit(n);
                if $|Q_{ACT} - Q_{EST}| > NTH$
                then
                    $Q_{TOT} \leftarrow Q_{TOT} -$
                    $ActQoS(INNER)$ ;
                    replan($INNER$, $Q_{TOT}$);
                end
            end
        case invocation
            $INNER \leftarrow node$
            $Q_{INNER} \leftarrow QoS(node)$;
            Execute(node);
            $Q_{INNERACT} \leftarrow ActQoS(node)$;
            $Q_{TOT} \leftarrow Q_{TOT} + Q_{INNERACT}$ ;
        if $|Q_{ACT} - Q_{EST}| > NTH$ then
            replan($INNER$, $Q_{TOT}$)
        end
    end
end function
```

Figure 5: Re-planning triggering algorithm

- a *Workflow QoS Optimizer*, to find the optimal set of concrete services for the workflow, with respect to the overall QoS, that maximizes a specified objective function and meets global QoS contraints. This component uses the *Concrete Workflow Builder* to obtain the concrete workflows needed for the evaluation of the objective function through the *Workflow QoS Estimator*, and an *Optimization Library* to solve the global optimization problem;

- a *Workflow Simulator*, for the service composition simulation. This component can be used to simulate the execution of a concrete workflow, to analyze the actual path followed and the actual local and global QoS values. When performing a simulation, we considered: i) the actual response times of the component services and the actual number of loop iterations, varying them according to a gaussian distribution function centered in the estimated value, and with a specified standard deviation; ii) the probability of choosing a case statement inside a switch construct, and the estimated availability of a service; iii) the values of the cost attributes of the component services are all constant, since it is unlikely (even if possible) that these could change at run–time. To avoid bias due to the result randomness, simulations are performed for a high number of times (say 1000), and average values of the actual QoS are considered. The *Workflow Simulator* has also been extended with the implementation of our Re-planning Triggering algorithm, and a *QoS Monitoring Tool*. Whenever a re-planning trigger occurs during execution, the slice of the workflow to be re-planned is computed and the *Workflow QoS Optimizer* is invoked to do the re-planning work on that slice. The *QoS Monitoring Tool* keeps track of the past workflow executions for each customer. The workflow log data is then used to refine the estimations on the loop iterations and to update the probabilities on case statements.

5. EMPIRICAL STUDY

The evaluation of the workflow QoS estimation and re-planning approaches has been performed through numerical simulation. The experiments were run on a *Compaq Proliant*^{TM} with Dual Xeon^{TM} 900 MHz processor, 2MB Cache and 4GB of RAM. To this aim, we used a simplified representation of the services, including a name, a reference to its semantic description and estimated values for cost, response time and availability attributes. The QoS values of semantically equivalent services were varying according to some gaussian distribution function, and better response time and availability offers corresponded to higher costs. Also, workflows of different sizes were generated with random probabilities on Switch and Loop iteration estimations.

Some sets of experiments were set up to reason about unlooping vs unfolding for a given workflow, when the estimated number of Loop iterations increases, and the frequency of re-planning during execution of workflows, due to deviations of the actual overall QoS from the estimated one. These experiments and the resulting data are discussed in the following subsections.

The GA was set up with the following parameters:

- Elitist GA, i.e., the best two individuals were kept alive over subsequent generations;
- Population of 100 individuals;
- Crossover probability=0.7; and

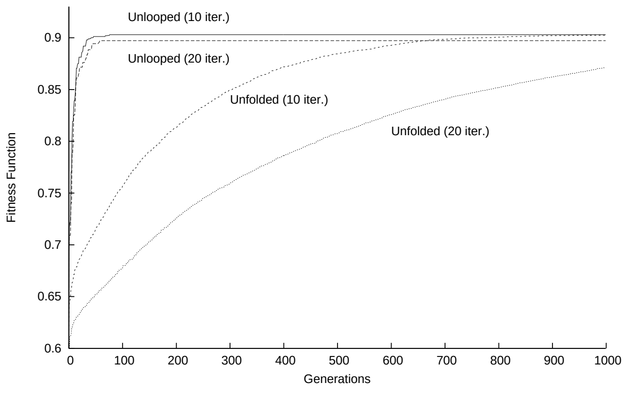


Figure 6: Unfolding vs. unlooping: fitness function convergence

- Mutation probability=0.1.

To avoid biasing results because of randomness, the GA executions were repeated 10 times, and average values used.

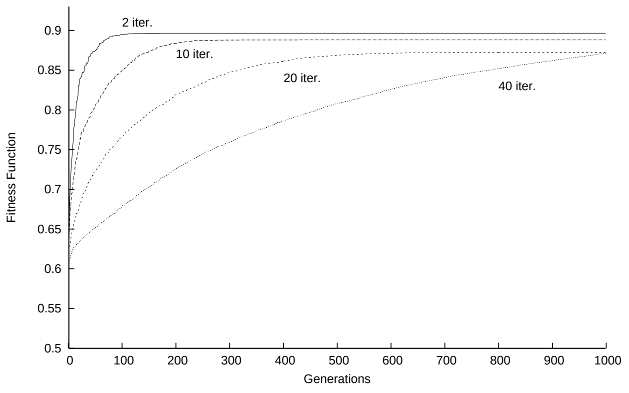


Figure 7: Unfolding rise time for different numbers of iterations

5.1 Unlooping vs. Unfolding

The first part of the experiments was concerned with testing both approaches for a benefits-costs analysis. We initially considered a simple workflow consisting of a Loop construct including 10 distinct abstract services, and repeated the experiments with a number of iterations ranged from 2 to 40, hence with 400 as a maximum number of service invocations. Also, for each abstract service, up to 15 concrete services were considered.

Figure 6 plots the fitness function evolution across GA generations, for worflows with 10 and 20 Loop iterations. It is important to note that without constraints, we get convergence to the same fitness value with both unfolding and unlooping approaches. However, the unlooping approach is able to ensure convergence to be reached much faster, after about 30-40 generations, regardless of the number of iterations considered. Vice–versa, as shown in Figure 8, the time rise of the unfolding case (i.e., the number of generations required for fitness convergence) depends on the number of iterations. Moreover, if we consider the GA execution performances [1] (see Figure 8), while those of the unfolding case are significantly higher (because of the largest genome) with an exponential grow at the increase of the number of iterations, the performances of the unlooping approach are lower and approximately constant with respect to the iterations.

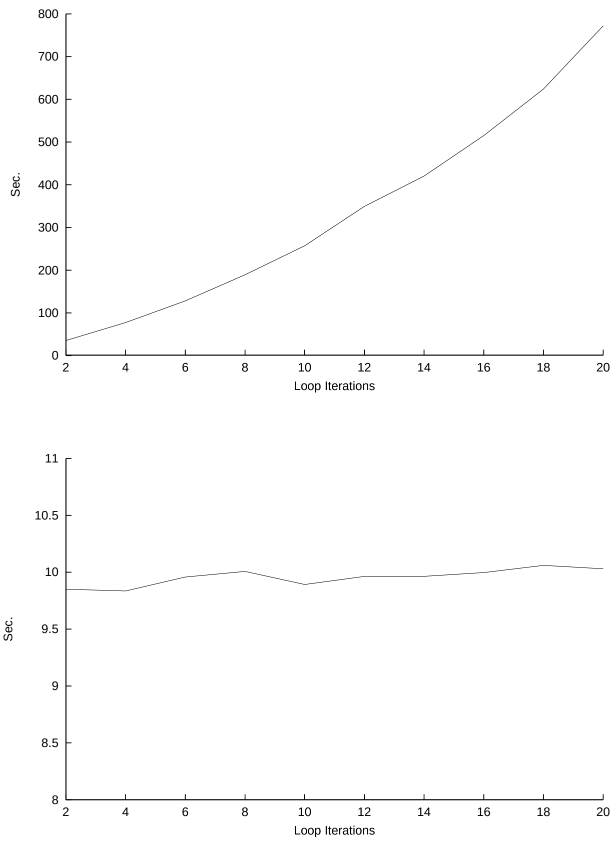


Figure 8: Unfolding vs. unlooping: timing comparison

5.2 Constraints

As described in Section 3, the unfolding approach may result useful in presence of global constraints. To support this hypothesis, we performed a second set of experiments, imposing constraints on the response time attribute (of type $overall\ response\ time \leq value$) , in order to evaluate the extent to which a loss of the overall quality with our unlooping approach, but with a time gain for finding an acceptable solution, is worth against absolute QoS optimality reached with the unfolding approach.

In particular, a workflow containing a Loop over 3 services was considered, with the number of iterations varying from 3 to 20. As shown in Table 2, the unfolding approach can lead to an increase of the fitness function value, although this is almost always limited by the maximum time to be taken. In our experiments, we found improvements of the fitness function values varying from 7 to 11%,

[1] *user* CPU times computed with the Unix utility `time`.

# of Loop Iterations	Fitness (Unlooped)	Fitness (Unfolded)	Fitness Increase (%)	Time (Unlooped)	Time (Unfolded)	Time Increase (%)
3	0.71	0.79	10.78%	2.80	8.47	202.50%
5	0.71	0.77	8.98%	2.81	12.63	349.47%
10	0.69	0.74	7.18%	2.84	26.84	845.07%
15	0.68	0.73	7.44%	2.87	43.28	1408.01%
20	0.67	0.72	8.29%	2.91	62.97	2064.66%

Table 2: Using the unfolded workflow in presence of constraints: cost/benefits

while the additional time needed for convergence was up to 2000%. Noticeably, varying the constraint proportionally to the number of iterations, we did not observe a corresponding increment in the difference between the fitness function values on the unfolded and unlooped workflows.

Thus, we believe that unfolding Loops may only be convenient when, in presence of constraints, even a slight improvement of the fitness value is worth hours taken to search for a solution. For example, we may desire to minimize the cost as much as possible, without trying to minimize the response time, on which, however, a constraint has been specified. Once the constraint is satisfied, the unfolding approach tends to choose the cheapest services rather than the fastest ones. On the contrary, the unlooping approach may require that each abstract service be bound to the fastest concrete service, since each abstract service is only bound once in the workflow. The tradeoff between the unfolding (better fitness) and unlooping (quick convergence to a sub-optinal solution) approaches is represented in Figure 9.

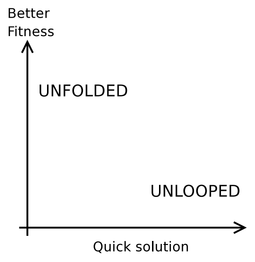


Figure 9: Tradeoff between unfolding and unlooping

5.3 Dealing with Uncertainty in QoS Estimation

The estimated overall QoS may deviate from the actual values obtained during service execution. The QoS of a Switch node is estimated, according to Table 1, as a weighted sum of the QoS values of the different cases. At run–time, only the QoS of the case actually followed is considered. Similarly, the actual number of Loop iterations can deviate from the estimated one. Finally, some QoS estimations of the invoked nodes can vary.

We used the *Workflow Simulator* to compute differences between estimated and actual QoS values. Simulations were performed varying from 10% to 50% the standard deviation on the estimated number of Loop iterations, and from 5% to 15% that on the QoS estimates (only for response time, since we considered cost values to be constant).

Figure 10 plots the error occurred on overall cost and response time estimates when the standard deviation on Loop iterations estimates

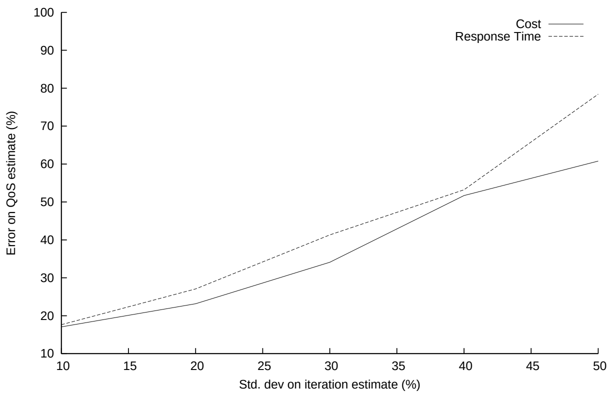


Figure 10: Error on QoS estimate due to uncertainty in the # of iterations

varies from 10 to 50. In this case the standard deviation on the QoS estimate was kept constant to 5%. Errors on availability were always below 1%, thus giving high confidence on service availability even in the presence of big estimation errors. When varying the standard deviation for the response time estimate, the error measured for 5%, 10% and 15% were, respectively, of 3%, 5.5% and 8%. In this experiment, the standard deviation on Loop iterations estimates was kept fixed to 1%.

The general indication given by the simulations we performed is that the actual QoS obtained during execution deviates in the presence of wrong QoS estimates and, above all, of wrong estimates on the paths to be followed in the workflow. This would require to: i) refine the estimates as much as possible, using actual data obtained during different service executions, and ii) trigger re-planning as soon as this is necessary.

5.4 Triggering Re-plan during Simulations

The last set of experiments aimed to simulate the behavior of the re-planning trigger during workflow executions. We considered two different workflows, composed of 10 and 12 nodes respectively, a standard deviation of 5% on QoS estimates, and of 10% on the number of iterations estimates. Finally, we calibrated the re-planning threshold to 10% (i.e., the difference between Q_{EST} Q_{ACT} needs to be bigger than 10%). In our experience (on different workflows, with the above specified estimate errors), thresholds bigger than 30% would not lead to any re-planning.

Figure 11-a shows that, for the first workflow, the re-planning likelihood is of 56.8%, distributed on different nodes (two Sequence, one Switch and one invoke node, indicated as $S18\#0$). In the second case (Figure 11-b), the likelihood of re-planning is of 45.6%, however limited to a unique, Switch node. This because the node splits the whole workflow in two cases with similar likelihood and, once the choice of the branch to be followed has been actually made, a re-planning is triggered to (eventually) re-plan only from that branch. It should be noted that the re-planning trigger can be useful not only at run-time, but also for analyzing how (and where) wrong QoS estimates could make a re-planning necessary during future executions thus wasting execution time.

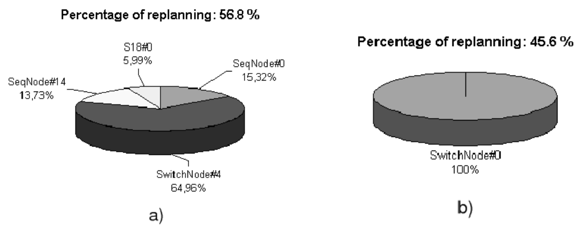


Figure 11: Triggering re-plan on different workflow nodes

6. CONCLUSIONS

In this paper we described an approach for QoS–aware service composition, based on composition of the QoS attributes of the component services and on Genetic Algorithms (GA). The aim of the approach is to provide a fast way, even if rough, to find the (sub)–optimal service composition and estimate its overall QoS. Constraints on the QoS attribute values are also kept into account. Obtaining an estimate quickly is particularly relevant for interactive services, where the time allowed to make the choice is limited. Numerical simulation showed the effectiveness of the approach and how an alternative approach, namely unfolding the workflow loops, can lead to better QoS values at the price of a higher search time. For example, the latter may be more suitable for the composition of non–interactive, computational intensive services.

In addition to the approach for composition, we proposed a re-planning algorithm to compute the deviation between the estimated QoS and the QoS measured at run–time, whenever possible. When the deviation goes above a threshold, the algorithm triggers a re-planning action, that is performed on the workflow slice that still remains to be executed.

Work–in–progress is devoted to better validate the approach on a large set of real services, as well as to further optimize the composition approach, for example with an hybrid optimization technique that combines GA and hill climbing. GA performances will also be compared with those of other optimization approaches. The whole toolkit is going to be integrated on a service broker we are developing in a project together with a large Italian software company.

7. REFERENCES

[1] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, S. T. D. Smith, I. Trickovic, and S. Weerawarana. Business process execution language for web services. *http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/*.

[2] J. Cardoso. *Quality of Service and Semantic Composition of Workflows*. PhD thesis, Univ. of Georgia, 2002.

[3] J. Cardoso, A. Sheth, J. Miller, J. Arnold, and K. Kochut. Quality of service for workflows and web service processes. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(3):281–308, April 2004.

[4] F. Casati, U. Dayal, D. Grigori, and M. Shan. Improving business process quality through exception understanding, prediction, and prevention. In *Proc. 27th International Conference on Very Large Data Bases (VLDB'01)*, pages 159–168, Rome, Italy, Sept. 2001.

[5] F. Casati and M. Shan. Dynamic and adaptive composition of e-services. *Information Systems*, 26(3):143–162, May 2001.

[6] K. Deb. Multi-objective genetic algorithms: Problem difficulties and construction of test problems. *Evolutionary Computation*, 7(3):205–230, 1999.

[7] M. Garey and D. Johnson. *Computers and Intractability: a Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979.

[8] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Pub Co, Jan 1989.

[9] ISO. *UNI EN ISO 8402 (Part of the ISO 9000 2002): Quality Vocabulary*.

[10] ITU. *Recommendation E.800 Quality of service and dependability vocabulary*.

[11] H. Ludwig. Web services qos: External slas and internal policies or: How do we deliver what we promise? In *Proc. 4th International Conference on Web Information Systems Engineering Workshops (WISEW'03)*. IEEE, 2004.

[12] H. Ludwig, A. Keller, A. Dan, R. King, and R. Franck. Web service level agreement (WSLA) language specification. *http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf*.

[13] M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara. Semantic matching of web services capabilities. In *Proceedings of the first International Semantic Web Conference (ISWC 2002)*, volume 2348 of *Lecture Notes on Computer Science*, pages 333–347. Springer-Verlag, June 2002.

[14] E. R. U.Greiner. Quality-oriented handling of exceptions in web-service-based cooperative processes. In *Proc. EAI-Workshop 2004 - Enterprise Application Integration*, pages 11–18. GITO-Verlag, 2004.

[15] K. P. V. Tosic, B. Pagurek. Wsol - a language for the formal specification of classes of service for web services. In *Proc. of the 2003 International Conference on Web Services (ICWS'03)*, pages 375–381. CSREA Press, 2003.

[16] W3C Working Group. Web services architecture. *http://www.w3.org/*.

[17] L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. Qos-aware middleware for web services composition. *IEEE Transactions on Software Engineering*, 30(5), May 2004.

A Framework for QoS-Aware Service Composition

Arnor Solberg
SINTEF
P.O. Box 124 Blindern
N-0314 Oslo, Norway
arnor.solberg@sintef.no

Sten Amundsen
Simula Research Labratory
P.O. Box 134
N-1325 Lysaker, Norway
stena@simula.no

Jan Øyvind Aagedal
SINTEF
P.O. Box 124 Blindern
N-0314 Oslo, Norway
Simula Research Laboratory
P.O. Box 134
N-1325 Lysaker, Norway
jan.aagedal@sintef.no

Frank Eliassen
Simula Research Labratory
P.O. Box 134
N-1325 Lysaker, Norway
frank@simula.no

ABSTRACT

Preparing an open environment for dynamic composition and re-composition of services requires standardized technologies for building, deploying, and running software systems. One key challenge in this respect is how to compose services and orchestrate the service collaboration to best fit the specified behavior, both in terms of functionality and quality. In this paper we present an approach for QoS-aware service composition. A general framework, called service planning framework, is presented. The framework is used at both build-time and run-time to identify possible implementations of a service and choose one service composition based on its QoS properties. At build-time we exploit model-driven system development, and at run-time we consider a QoS-aware execution environment.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management – *Software quality assurance, productivity.*

General Terms

Management, Design.

Keywords

Service composition, QoS, modeling, specification, MDA, model-transformation, QoS-aware adaptation.

1. INTRODUCTION

Business rely more and more on distributed computing systems for collaboration and trading between businesses and between businesses and customers, i.e., Business to Business (B2B) and Business to Customer (B2C). In most cases, business is dynamic, so work methods and processes evolve over time. Thus, the supporting systems also need to act dynamically and evolve in pace with changing environments and requirements. In addition, a distributed system needs to cope with different technologies and continuous changes in the execution environment (due to constrained system resource availability, load, etc.). One appealing way of meeting these challenges is to dynamically derive the structure and implementation of the system on demand. Hence, based on specified requirements and constraints, the requested service(s) are composed dynamically, e.g., by orchestrating a set of available autonomous software entities (sub-services). In this context, a significant aspect is the provided quality of service (QoS). Services providing the right quality to the right price will be requested. Then the challenge remains to compose the service(s) based on specified requirements and price constraints, utilizing available environment resources and services.

To handle the complexities of distributed systems, object oriented technology was an important step forward. It improved the separation of concern between the different software entities in the system. Currently system developers have embraced components as the most suitable software entity for designing and developing distributed systems. Component technology includes important principles like encapsulation, interfaces, assumptions (i.e., required interfaces) and reflection/introspection. From a user perspective, a component-based system may be seen to offer a set of services. These services are typically provided by a composition of collaborating components. Due to the encapsulation property, we might compose new services from existing services, i.e., a recursive service composition. This view of systems and the idea of late bindings are key aspects of the Service Oriented Computing (SOC) paradigm [3].

The focus within component-based software engineering has mainly been on modeling the functional properties [2], and developing suitable execution environments for publishing and running services. However, composing services on demand for B2B and B2C systems (e.g., in a global web environment using the Web service approach [4]), consideration of QoS (e.g., cost, availability, execution delay, reputation and successful execution rate) is considered to be vital. To gain proper management and to be able to offer services with QoS-guarantees, we advocate that QoS constraints should be carefully considered during the development phases and also managed during execution.

This paper presents a general framework for QoS-aware service composition (section 2). The framework includes a concept model defining the core concepts for QoS-aware service composition, and the specification of the behavioral aspects of what we have denoted service planning. Application of the framework is described in section 3. This section describes how to apply the framework at build-time using a model-driven approach, in alignment with the model-driven architecture (MDA™) philosophy [7], as well as how the framework might be applied at run-time for planning service composition and re-composition. Build-time QoS modeling is based on the current version of the

UML profile for QoS [8]. At run-time the framework assumes an execution environment that implements reflection [14], and uses run-time reconfiguration mechanisms of component compositions [10][11]. A video-conference system is used to illustrate how the framework is applied at both build- and run-time.

2. SERVICE PLANNING FRAMEWORK

The service planning framework defines a concept model and the behavioral aspects of the service planning activity.

2.1 Concept Model

The concept model is shown in Figure 1. The three main concepts are: 1) service type, 2) service plan, and 3) blueprint. The service type represents the service properties and is independent of the implementation. The service plan specifies the service composition and the QoS-profile of a service type. A blueprint realizes a service type according to its associated service plan. The concepts of the framework are described in more detail in the following paragraphs.

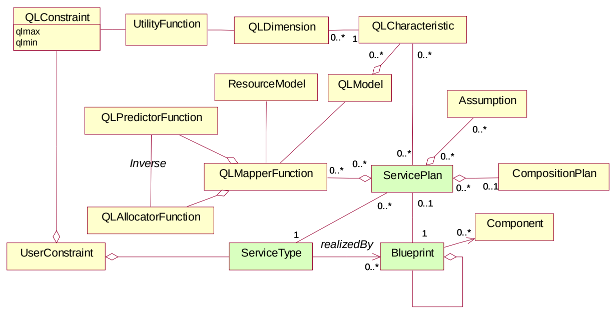


Figure 1. The framework concept model

ServiceType; Defines the service name and the provided functional services in the form of operation signatures and semantic descriptions. There are different ways to represent a service type, like web service description language (WSDL), OMG's interface definition language (IDL) and Java interface description language (JIDL), even if these do not support semantic descriptions.

Blueprint; A blueprint is a persistent immutable value, which realizes the service type. There can be many possible realizations of a service type, each of which can have different QoS profiles (described in the associated service plan). A blueprint encompasses a recursive structure, thus, a blueprint can enclose other blueprints. Its representation will typically vary with the abstraction level. At the model level a blueprint can be represented as a UML design model, either at a platform-independent level (PIM-level) or at a platform-specific level (PSM-level). At the platform-specific level, blueprints may also be represented in code (e.g., EJB jar files containing compiled Java classes).

ServicePlan; Contains information elements specifying how to compose the service from blueprints (the *CompositionPlan*), and which QoS-properties this composition will have (*QL-Characteristics*). A service plan may typically be specified in the eXtensible Markup Language (XML), and if the associated blueprint is a UML-model, the service plan is partly represented using XML Metadata Interchange (XMI). A service plan specifies what platform the blueprint is designed for, which may be a UML-platform or a middleware platform. At the UML-level one may have either the pure UML-platform (the UML meta-model) or a specific UML profile, like the UML EJB-profile [16]. Since a service plan denotes the actual platform of the blueprint, there is one service plan for each blueprint. This also implies that you will have a recursive structure of service plans in accordance with the recursive structure of the blueprints. The information elements in a service plan are:

- *Assumption;* A list with service configuration data, platform requirements and dependencies to the associated platform. The platform requirements specify the platform where the blueprint can be interpreted, for instance a UML-platform or a QoS-aware execution environment.
- *CompositionPlan;* Specifies the service composition, i.e., the composite service types and bindings between them. Figure 2 illustrates how composition plans specify the set of service types in a composition. Note that for an atomic service (or an atomic blueprint), the composition plan of the associated service plan is empty (it returns *atomic*).

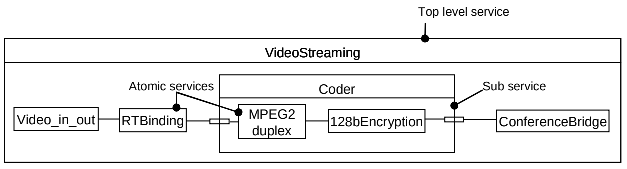


Figure 2. Service types at different levels in the composition

- *QL-Characteristics;* A set of domain quality loss characteristics (QL-Characteristics) constitutes the quality loss model. Each QL-characteristic is specified through a set of quality loss dimensions. The QL-characteristic is used to specify QoS-properties of the service composition (in the composition plan), using the QoS semantics described in [21]. It measures quality relative to perfect quality, i.e. the difference between perfect and achieved quality. This difference is called *quality-loss (QL)* [13]. QoS, on the other hand, is viewed as the difference between minimum achievable quality and actual quality. Figure 3 illustrate the QL-measure and QoS.

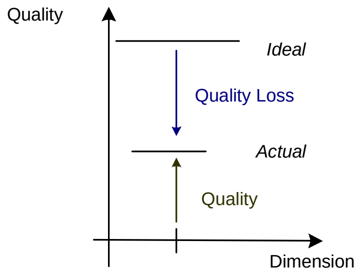


Figure 3. QL versus QoS

Formally, QL is the difference between ideal and actual output traces, where a trace is the output messages from a service. The ideal output trace, *I*, is defined to be the service output when system resource availability is infinite. Actual output trace, *A*, is defined as the service output with finite and shared system resources. The difference between the ideal and actual is a series of vectors. Figure 4 illustrates the two output traces and the difference vectors, *C*, between them for *n* output messages. The figure shows how the time of the output event and the value of message content for each actual output message differ from the ideal. The number of ways in which an actual trace may differ from the ideal trace may explode as the complexity of the message structure increases. Fortunately, we frequently are concerned only with an overall measure of distance from the ideal. We therefore define an error model as a set of QL-functions over the series of input vectors. Each QL-function is typically defined as an aggregating statistical measure such as maximum, mean value or variance This approach makes QL-functions useful and suitable for computations.

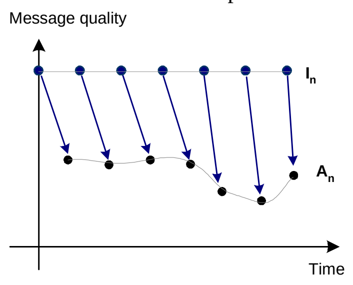


Figure 4. Output message traces and difference vectors

To define ideal and actual output trace for a service, one relates the output message to the service input message. *Ideal* is achieved when the service is computing indefinitely fast, which is zero delay when considering the dimension delay. Hence, if a service output happens at time *t* the related service input would have happened at time *t* as well. Figure 5 illustrate the casual relationship between service output and input, which is used to define the QL-dimension and the QL-function.

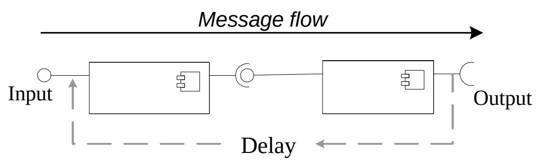


Figure 5. The input-output relationship for the QL-dimension delay

- *Utility function;* To capture user constraints, the concept model uses what is called QL-constraints. Users specify their constraints as minimum and maximum acceptable QL, ql_{min} and ql_{max}, for each QL-dimension. The usability of the service is represented by the utility measure, a real number in the range [0..1]. Utility is expressed by a utility function, which is *a function that maps dimensional quality-loss in a single quality dimension to utility values* [12]. Hence, for each QL-dimension there is one utility function. Figure 6 shows one possible utility function together with the associated QL-constraints. The utility function enables the framework to identify the service plan that; 1) meets the user constraints and 2) optimize the usability. You can also derive aggregate utility functions based on existing utility functions, to get utility values for a combination of quality dimensions, e.g. media quality including both audio and video quality.
- *QL-MappingFunction;* There are two types of mapping functions; *QL-prediction (QL-PredictorFunction)* and *QL-allocation (QL-AllocatorFunction)*. The prediction functions encode the application developer's knowledge about the service, and predict the quality loss for the service composition as a function of composition and system resource availability (CPU, disk, network, etc). Prediction functions can also be made recursive, by invoking appropriate prediction functions in each sub-service plan. The QL-allocation function budgets the quality loss down to each sub-service and atomic service in the service composition. Input is the QL- limits, ql_{min} and ql_{max} for the QL-dimension of interest. At leaf-level the QL-allocation functions define the system resource requirements, which must be expressed in a form that can be understood by the resource manager.

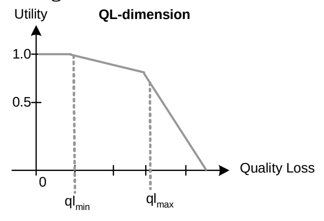


Figure 6. Utility function and ql_{min} - ql_{max}

2.2 Behavior Model

The behavior model specifies the behavioral aspects of the service planning framework. It includes a set of active elements as shown in Figure 7.

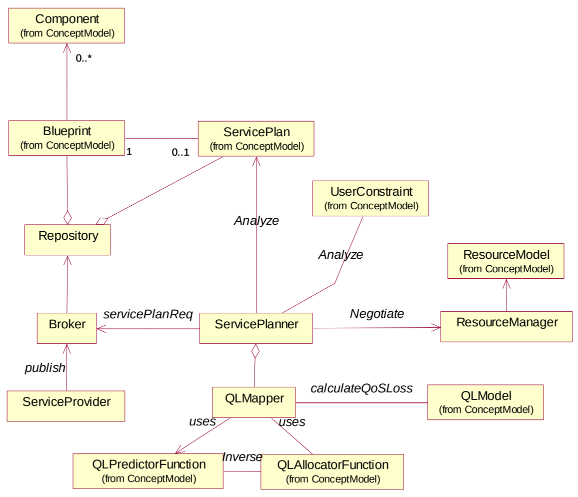

Component (from ConceptModel)
0..*
Blueprint (from ConceptModel)
1
0..1
ServicePlan (from ConceptModel)
UserConstraint (from ConceptModel)
Repository
Analyze
Analyze
ResourceModel (from ConceptModel)
Broker
servicePlanReq
ServicePlanner
Negotiate
ResourceManager
publish
ServiceProvider
QLMapper
calculateQoSLoss
QLModel (from ConceptModel)
uses
uses
QLPredictorFunction (from ConceptModel)
Inverse
QLAllocatorFunction (from ConceptModel)

Figure 7. The framework behavior architecture model

The active elements in the framework are: the service provider, broker, service planner, QL-Mapper, and resource manager. They interact to: 1) publish alternative service compositions of service types and 2) identify and choose a service composition that meets the users QL-constraints.

ServiceProvider; The actor (organization, system architects, application developers, etc.) that has a service to publish for others to use. A service provider interacts with the broker to deploy blueprints and service plans, and to add service types.

Broker; a discovery service similar to the CORBA trading service [18] or the Universal Description, Discovery and Integration (UDDI) [19], where blueprints and service plans can be published and discovered according to specified properties. Service providers advertise their service types and associated service plans, enabling the broker to map between a service type and alternative service plans deployed for the service type. The broker relies on a repository for storing/retrieving service plans and blueprints.

ServicePlanner; Identifies implementations of a service type that guarantee correct functional behavior according to the type and meets the specified QL-constraints. The service planner uses the broker to discover alternative service plans for a service type, and it uses the resource manager for negotiating system resources. This responsibility includes validating that assumptions in the service plan are met, and predict QL for each alternative service composition. This is coordinated by the QL-mapper which can read and analyze the QL-characteristics and QL-mapping functions *(QLPredictorFunction* and *QLAllocatorFunction*) in the service plan. The service planner can be set up to return one or all service compositions that meet the user's constraints. If none is found it rejects the service request from the user.

Resource Manager; Uses the resource model to retrieve characteristics and availability of system resources in the platform. When the framework is implemented in a QoS-aware execution environment, the resource manager also supports resource negotiation and monitoring.

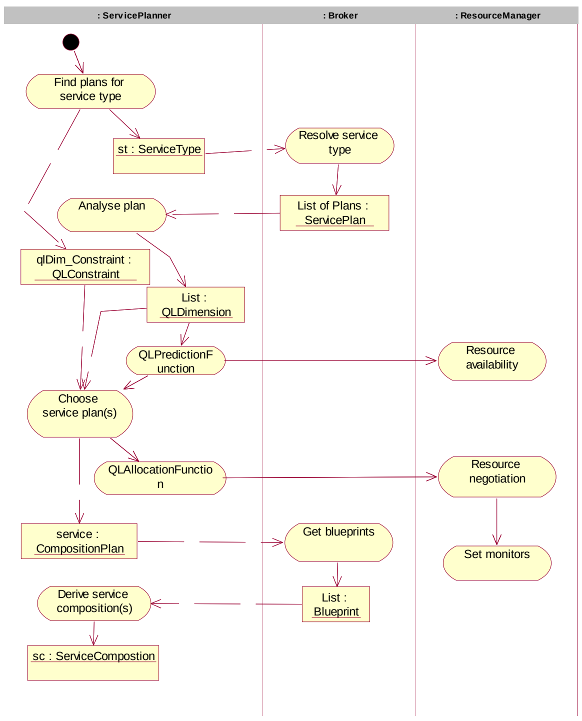


Figure 8. Service planning activities

The active elements in the service planning framework perform two main tasks: 1) store provided blueprints and associated service plans for a service type as requested by the ServiceProvider, and 2) identify and choose service compositions(s) that meet functional and QoS-constraints when receiving a service request. Figure 8 illustrate task 2) in a UML activity diagram. The required stimulus is a service request that includes the service type and the associated QL-constraints. The broker comes up with a list of alternative service plans for the actual service type. The service planner selects the appropriate alternatives from this list, through analyzing QL-characteristics and user defined QL-constraints. It then performs QL-prediction and QL-allocation and negotiates with the resource manager in order to allocate the required resources. If it fails to allocate required resources, the plan will be eliminated from the list. The set of composition plans and associated blueprints are used in order to derive the appropriate service compositions as output.

3. APPLYING THE FRAMEWORK

In this section we describe how the service planning framework can be applied build-time and run-time. At build-time we employ a model-driven system development approach, based on the MDA [7] philosophy, where the service planning framework is used to perform QoS-aware model transformations.

When developing a system one typically consider a broad set of possible quality and functionality requirements during the inception and early elaboration phases. Different kinds of more or less crucial functionality are elaborated, and what QoS characteristics to consider is identified. However, the exact values of the QoS characteristics are typically not settled at this stage. During late elaboration and construction phases the requirements to be implemented for this version of the system is determined. Some requirements are typically modified, QoS values and ranges are resolved and some requirements may be ignored (e.g., to shorten the time to market or to aim at a specific market segment). Thus, a more extensive set of possible solutions is typically considered at the higher abstraction levels, while at the lower abstraction levels more decisions are made, reducing the set of possible variations. Consequently, a selection of possible compositions for a service is made during model transformations (e.g., for PIM to PSM transformations). Another consideration in this respect is that the PIM abstraction level makes the solutions independent of existing and upcoming platforms and versions of platforms. Thus, covering a broader set of variations at the PIM level, and resolve variability through a PIM to PSM transformation will provide an efficient system development process. Different decisions might apply for different platforms in order to exploit differences in platform capabilities.

At the execution level one may still have a set of possible solutions to choose between. The selection is then performed dynamically, at run-time, preferably in a client transparent manner.

The concepts of the system planning framework are used both during model transformation and during system execution to perform QoS-aware reasoning and to choose alternative service compositions.

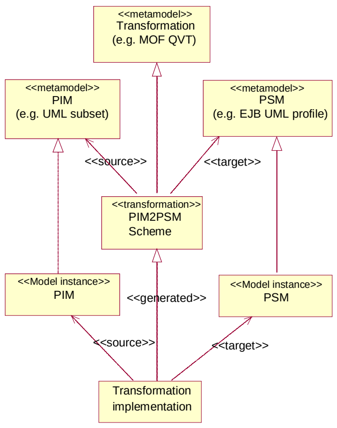
At run-time the service planning framework is designed to be implemented in a reflective execution environment, or as a dedicated management service that offers management and introspection of executing service compositions by means of their associated service plans. This is then utilized to provide mechanisms for transparent configuration and reconfiguration of service compositions.

Part of our research within the area of QoS-management is designing a new reflective component architecture, with a small core. It is called QuA - QUality of service aware component Architecture [15], currently available in two prototypes, one implemented Smalltalk and the other in Java. It has been developed for prototyping new ideas and concepts within the area of dynamic QoS management. The service planning framework is one result from this work. Parts of the service planning framework have already been implemented in the QuA prototypes.

3.1 MDA and Model Transformation

In model-driven system development, an extensive set of interrelated models at different abstraction levels are developed. The key challenge is to define, manage, and maintain traces and relationships between different models, model views and model elements, including the code of the system. An advanced MDA-based framework should provide well-structured support for modeling at different abstraction levels, and be able to automatically perform roundtrip model transformations as well as code generation.

Model transformation can be viewed as a transformation between two model spaces defined by their respective meta-models. Thus, transforming a PIM to PSM is achieved by a generic transformation specification, which specifies how a meta-model concept of the source model (PIM) should appear in the target model (PSM). The transformation specification itself is also according to a meta-model defining the transformation specification constructs. This is illustrated in Figure 9.



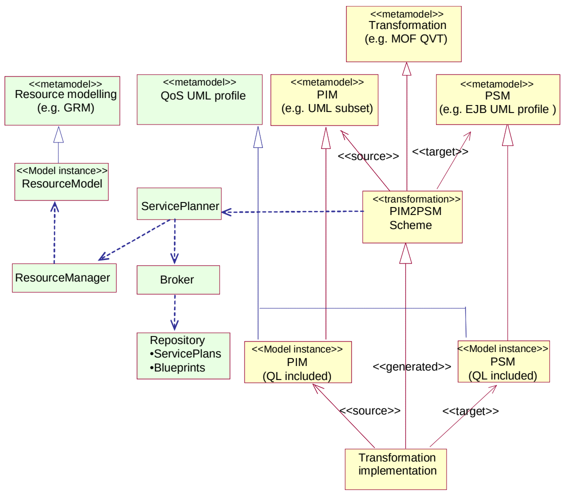
<<metamodel>> Transformation (e.g. MOF QVT)
<<metamodel>> PIM (e.g. UML subset)
<<metamodel>> PSM (e.g. EJB UML profile)
<<source>>
<<target>>
<<transformation>> PIM2PSM Scheme
<<Model instance>> PIM
<<generated>>
<<Model instance>> PSM
<<source>>
<<target>>
Transformation implementation

Figure 9. Conceptual transformation model

To make the model transformation QoS-aware, the QoS-properties needs to be integrated into the models. QoS-properties include significant information to enable extensive model transformations, and, more importantly, to deliver efficient code. Thus, it is important to understand the implications of the QoS-requirements and specifications when performing model transformations in order to deliver high-quality results.

Figure 10 shows how we utilize the service planning framework in order to accomplish QoS-aware model transformations.

The transformation is exemplified with a PIM to PSM transformation. The source model is a PIM model including PIM-level QL-specifications in alignment with the UML-profile for QoS [8]. At the model level, the service plans typically describe design patterns. The blueprints are the actual realizations of the patterns in the form of models according to the target platform (in the MDA context this will be the UML-platform). The service plan encapsulates the meta-data including the QL-characteristics of a specific blueprint. The resource manager uses a platform specific resource model, which is the specification of available resources of the target platform, to negotiate resources with the service planner, e.g., to deliver efficient deployment models. The resource model instance is according to a resource modeling meta-model, which could be based on the General Resource Model (GRM) described in [15]. The framework behavior in QoS-aware model transformation is further elaborated in section 3.3



<<metamodel>> Transformation (e.g. MOF QVT)
<<metamodel>> Resource modelling (e.g. GRM)
<<metamodel>> QoS UML profile
<<metamodel>> PIM (e.g. UML subset)
<<metamodel>> PSM (e.g. EJB UML profile)
<<Model instance>> ResourceModel
<<source>>
<<target>>
ServicePlanner
<<transformation>> PIM2PSM Scheme
ResourceManager
Broker
Repository •ServicePlans •Blueprints
<<Model instance>> PIM (QL included)
<<generated>>
<<Model instance>> PSM (QL included)
<<source>>
<<target>>
Transformation implementation

Figure 10. QoS-aware model transformation

3.2 Case Description

Composing services on demand requires provision of autonomous services from which to select, configure and compose. For instance, in a Web-service environment there will be service providers publishing autonomous Web-services providing specific functionality at certain quality and price. A service provider typically wants to provide different compositions and configurations of each provided service type to cover different quality demands of an actual service, to be able to serve a wider set of users. A Web-service may be composed of sub-services controlled by the actual service provider (*internal service composition*) or it can be composed partly or totally of other public available web services (*external Web-service composition*).

To illustrate the service planning framework, we only consider internal service planning. However, the framework can also be used for external service planning, like creating a Web-service composition from existing Web-services. For run-time QoS-aware service compositions, the service planning framework may then be published as a Web-service that configures and reconfigures the Web-service composition as indicated in Figure 11. We believe that end-to-end QoS-guarantees can only be provided by a combination of external and internal service planning.

A video conference service is used to illustrate how the proposed service planning framework makes the service compositions QoS-aware. The system has a centralized video conference server, which advertises its service as a Web-service by means of a UDDI server. Figure 11 depicts the system.

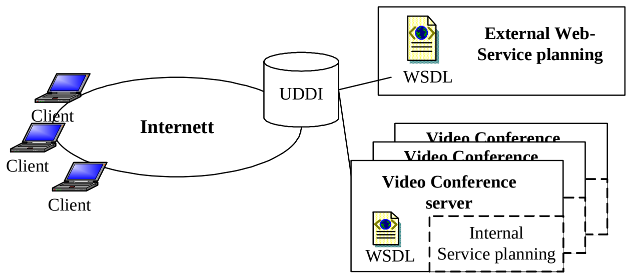


Figure 11. Video conference system overview

An end-user of a video conference service typically has QoS requirements with respect to maximum acceptable delay and minimum acceptable video quality. When the client is using wired based networks to access the server, these requirements can easily be met. But when wireless is used, the data rate will impose restrictions that can only be solved by using video coders designed for wireless networks. This means that there are alternative solutions with different QoS-properties. Furthermore, end-users are often using laptops. This allows them to move during a video conference session. Thus, the system must at run-time choose the video coder that can meet the end-users QoS-requirements.

Distributed systems, like video conferencing, may have service composition that include the clients. For these systems the framework requires an execution environment on the client side, which supports 1) downloading of blueprints from a remote repository, 2) dynamic loading of blueprints and instantiation of components, 3) binding components together, and 4) reflection mechanisms accessible for a remote service planner.

3.3 QoS-Aware Model Transformation

A PIM of the Video conference service using UML 2.0 [19] composite structure is shown in Figure 12.

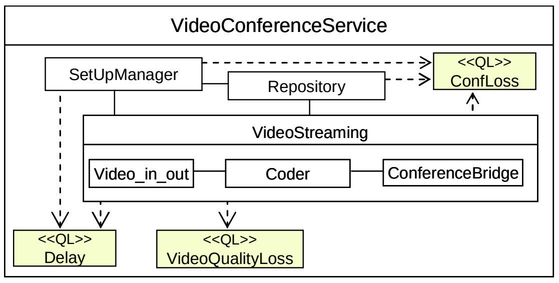


Figure 12 PIM model including QoS

The video conference service is composed of: setup manager, repository and video streaming service. The setup manager is responsible for initializing and setting up video conferences on request from end-users. There is a repository for storing management data like metadata of video conference sessions. The video streaming component is responsible for the streaming video to the clients. There is associated a set of QL-characteristics by means of dependencies to <<QL>> stereotypes. The QL stereotype of UML class is assumed to be defined as part of the PIM profile. The QL-characteristic of concern for this application domain is defined to be confidentiality loss (*ConfLoss*), *Delay* and *Video QualityLoss*. The QL-characteristics will typically be defined at a proper domain level, and will form the QL-model in concert with the QL-mapper functions (as depicted in Figure 1). The QL-characteristics are defined using the UML profile for QoS [8].

Figure 13 show the quality loss characteristics and dimensions for the video conference service, together with the specified units and allowed values. Ideal output traces for the QL-dimensions are: 1) output confidentiality equals input confidentiality with respect to message protection, authorization and data protection dimensions, 2) zero delay, 3) output message rate equals input message rate, 4) output message is a correct representation of the input messages, and 5) output image resolution equals input image resolution.

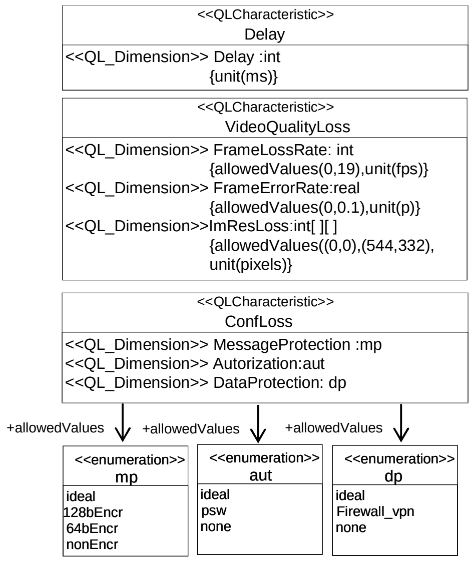

<<QLCharacteristic>> Delay
<<QL_Dimension>> Delay :int {unit(ms)}
<<QLCharacteristic>> VideoQualityLoss
<<QL_Dimension>> FrameLossRate: int {allowedValues(0,19),unit(fps)}
<<QL_Dimension>> FrameErrorRate:real {allowedValues(0,0.1),unit(p)}
<<QL_Dimension>>ImResLoss:int[ ][ ] {allowedValues((0,0),(544,332), unit(pixels)}
<<QLCharacteristic>> ConfLoss
<<QL_Dimension>> MessageProtection :mp
<<QL_Dimension>> Autorization:aut
<<QL_Dimension>> DataProtection: dp
+allowedValues
+allowedValues
+allowedValues
<<enumeration>> mp
ideal
128bEncr
64bEncr
nonEncr
<<enumeration>> aut
ideal
psw
none
<<enumeration>> dp
ideal
Firewall_vpn
none

Figure 13 Definition of the QL-characteristics

Using the description of ideal and actual output traces as illustrated in Figure 5, one can define the QL-functions for each QL-dimension. The delay QL-dimension is defined to be an integer specifying latency in milliseconds. The domain-specific video quality loss characteristic is defined to have three QL-dimensions; frame loss, frame error rate and image resolution loss. The specified *allowed values* for these QL dimensions are: for frame rate loss 0 to 19 frames per second (fps), for frame error rate a probability measure from 0.0 to 0.1 and for image resolution loss from (0*0) to (544*332) samples. Confidentiality is the ability of a system to restrict access to information to authorized users only. The QL-dimensions defined for confidentiality is message protection, authorization and data protection. The enumerations shown in Figure 13 define the allowed values. Then utility functions are derived for each QL-dimension. The utility function for message protection is shown in

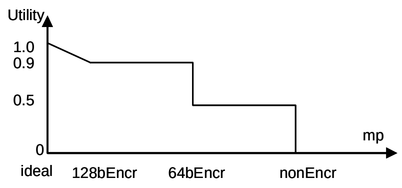


Figure 14 Utility function for message protection QL-dimension

When performing a model transformation the QL-dimensions can be partly or completely resolved by specifying the *qlmin* and *qlmax* of the QL-dimensions. This specification is denoted as QL-Constraints in Figure 1. To specify an exact value one sets *qlmin*=*qlmax*=value. In the following we perform a model transformation from PIM to PSM where the following QL-constraints are specified for the video streaming service: *ConfLoss*(*qlmin*=*qlmax*=*nonEncr*), *FrameLossRate*(*qlmin*=0, *qlmax*=10), *ImRes*(*qlmin*=(0,0), *qlmax*=(512,512)) and *Delay*(*qlmin*=0, *qlmax*=200). Constraints on the QL-dimension *FrameErrorRate* (defining at which level bit errors are corrected) is not specified, and thus, not accounted for in this transformation.

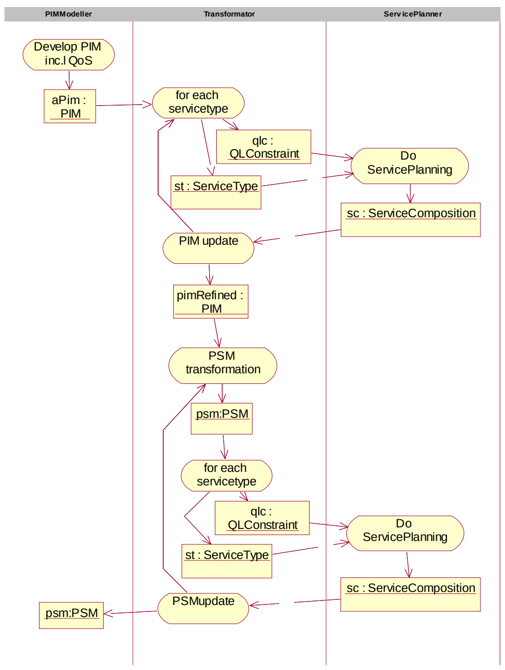


Figure 15. PIM to PSM transformation

Applying the service planning framework, the transformation process will perform as depicted in Figure 15. The PIM model and QL-dimensions with associated specifications of values and range are input to the transformator. The transformator uses the service planning framework to get appropriate service compositions according to the QoS requirements. Note that this is performed both at the source level (PIM) and at the target level (PSM). The service planner utilizes the utility functions to map QL-Constraints to utility values. The service planner performs according to the UML activity diagram in Figure 8. At the PIM level the service planner uses the broker to get the list of appropriate PIM level service plans with associated blueprints (if any). In this example, there is only published alternative service compositions related to the message protection QL-dimension . These are shown in Figure 16.

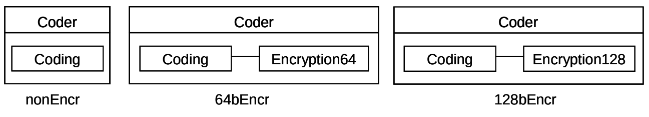


Figure 16. Coder Composition plans

We see that the coder service type has three different composition plans, each of which has different security properties (nonEncr, 64bEncr, and 128bEncr). Since *qlmax*=nonEncr in our example the nonEncr coder will be selected. The service planning process is used to select the appropriate composition in the form of a service plan with associated blueprint. When the appropriate selection is done the transformer integrates the service composition blueprint into the PIM model and derives the PSM of the Video conference service (e.g., towards the EJB platform). Then the transformator again utilizes the service planner framework to get hold of alternative PSM-level compositions according to the specified requirements. In our example it comes up with three possible PSM-level compositions for the videoStreaming service which are listed in Figure 17.

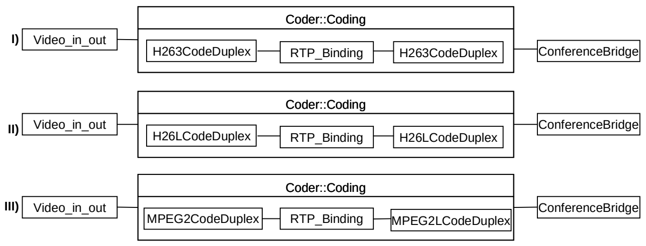


Figure 17. Three alternative composition plans

QL-prediction functions map the availability of system resources to QL along each QL-dimension. System resources considered relevant for the video conference system are: memory, CPU, and network capacity. During build-time application developers are free to decide their own format and the complexity of the error prediction functions, hence, the functions may be a straightforward condition statement or a complex description measured quality loss. Composition I of Figure 17 is used to illustrate one possible implementation of a QL-prediction function. The QL-prediction function takes available memory, CPU, and network capacity as input, and returns result set (RS) a or b (see table 1).

Table 1 shows the assumptions, QL-dimensions, and QL-prediction functions for composition I. The QL-allocation function is not shown in table, since it is the inverse of the prediction function.

We now assume that all these alternatives are deployed and published to the run-time service planning. The following subsections will then explore how the service planning framework is applied to perform QoS aware service composition dynamically at run-time.

3.4 Deployment and Service Request

Blueprints are stored together with associated service plans in a logically global repository. In the video conference case, this means that one deploys service plans and blueprints for the alternative service compositions. The service, *VideoConference*, can then be published in a directory server, such as a UDDI and CORBA name/trader. From the directory server, end-users get information about the published service, including utility functions. End-user specifies QoS requirements by setting minimum and maximum QL values in each dimensional utility function. Figure 18 illustrates one of the utility functions for the *VideoConference* service. In the figure are the end-users QoS-requirements shown; minimum QL is set to zero and maximum

Plan Properties	Composition I
Assumptions:	
- I/P format	4:2:2 YCRCB(ITU-R 601 –PAL)
- I/P frame rate	[20 – 50]
- Platform	Java
QLModel:	
- MessageProtection	Ideal, 128bEncr, 64bEncr, nonEncr
- Delay	0 – 2 s
- Frame loss rate	0 – 19 fps
- Frame error rate	0.0 – 1.0 p
- Image resolution	[0*0, …. 544*332] pixels
QLPrediction	

RS = (condition) ? [a], [b]

$$\begin{bmatrix} \text{MessageProtect} \\ \text{Delay} \\ \text{Frame loss rate} \\ \text{Frame error rate} \\ \text{Image resolution} \end{bmatrix} = \begin{bmatrix} MEMclient > 20kB \\ MEMserver > 1MB \\ CPUclient > 200MIPS \\ CPUserver > 700MIPS \\ NET > 9kbit/s \end{bmatrix} ? \begin{bmatrix} nonEncr \\ 0{,}14 \\ 0 \\ 0{,}015 \\ 368*183 \end{bmatrix}, \begin{bmatrix} nonEncr \\ 2 \\ 19 \\ 0{,}015 \\ 544*332 \end{bmatrix}$$

Table 1. Service plan for composition I

QL set to 9 fps. The end-user then sends a service request, with service type and utility functions, to the server.

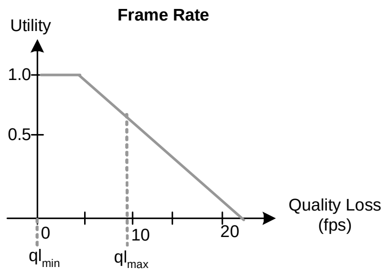


Figure 18. User QoS-requirements for one QL-dimension

3.5 Service Instantiation

A service request from an end-user specifies the service type and the user QoS-requirements. In distributed systems, like B2B and B2C, the network data rate has a major impact on experienced QoS. End users connected to a LAN have available a data rate that allow for larger chunks of data. For run-time service planning it is important that the resource models, there are more than one, are updated with the data rate available. Hence, it can choose a composition that both meets the users QoS-requirements and maximize utility. When the service planner receives the service request, it uses the broker to find the service plan. This plan specifies the overall composition at a logical level using service types. For each service type specified in the service plan, the service planner searches for assocoated service plans and blueprints. Table 1 shows the service plan for composition I, but without the composition plan

First the service planner checks that the assumptions in the plan can be met. Then it uses the QL-model to identify QL-dimensions, acceptable values and units (expressed by the QL-function). Finally the service planner predicts the QL using the prediction function specified in the service plans. For the video conference service, there are three alternative compositions. After predicting the QL for these three, the service planner uses the utility functions to compare the predicted QL against the user QoS requirements. For a video conference with high data rate and low load on both client and terminal, all three compositions meet the end-users requirements, i.e., between ql_{min} and ql_{max} as shown in Figure 18 for the frame rate QL-dimension. The planner then chose the composition that gives the highest utility, which in our case is composition III. When accessing the video conference server over LAN, the composition using MPEG-2 video coding has the lowest QL along image resolution giving higher utility, and acceptable QL along the other four QL-dimensions.

Finally, the service planner uses the QL-allocation functions to calculate the system resource requirements. The resulting resource vector is forwarded to the resource manager for reservation and monitoring. The middleware then dynamically loads the blueprints, instantiate the components, and bind them together to form the service.

3.6 Service Adaptation

After some time, the end-user disconnects the Ethernet cable and move over to WLAN. It is assumed that network connection and the RTP session are successfully re-established on the new network. After a short time period, the resource monitors detect the reduction in network capacity, and notify the service planner about the change. The service planner now needs to re-composition the service to maintain the committed QoS-level. Figure 19 shows the interaction the between the service planner and the other active elements in the service planning framework.

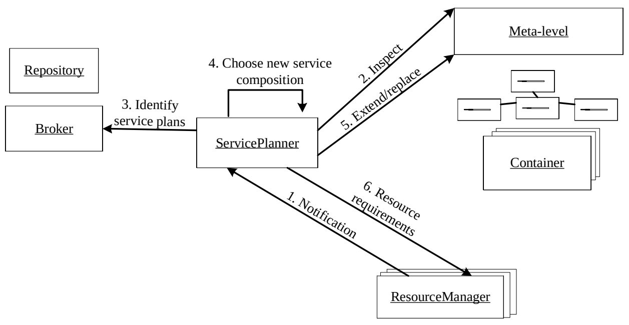


Figure 19. Behavior during service adaptation

After being notified, the service planner inspects the running service composition using reflection. Then it resolves the service type *VideoConference* and receives the service plan from the broker. For each service type in the plan the service planner resolves all the way down to the blueprints, giving a hierarchy of service types and service plans. The planner compares alternative service compositions, as it did when choosing the initial service composition. Predicting the QL shows that the low data rate in WLAN increases the QL.

For composition III, which uses MPEG-2 video coding, the QL is too high along delay, frame loss rate and image resolution loss. Composition I and II, on the other hand, meet the user requirements. To choose between the two compositions, the service planner uses the utility measure. Composition II has an advantage over composition I since it uses a H.263 coder with inbuilt FEC. This gives the lowest QL and highest utility along the frame error rate dimension. Using reflection, the service planner stops the two MPEG code/decode components, *MPEG2CodeDuplex*, and replaces it with *H26LCodeDuplex*. The resource demand is recalculated by using the QL-allocation to the service plan for composition II. Finally the resource vector is forwarded to the resource manager(s) for reservation and monitoring.

4. RELATED WORK

In this section we discuss related work within the areas QoS-aware service compositions, model driven QoS-aware transformation and component based execution environments.

In [28], a QoS compiler is presented that translates user-perceived QoS levels to a run-time ”script” corresponding to our service planning concept. It is assumed that application developers can provide a QoS-specification with knowledge of system-level QoS representations. This approach appears to yield an application that can only be deployed with the particular set of QoS-management services understood by the application developer. Another approach, [29], addresses QoS-awareness by capturing non-functional properties in a QoS-model, specifying service compositions as task graphs. It selects services based on QoS criteria, and employs an adaptive execution engine that re-plans the composition of services. Their ontologies and implementation of local and global service planning, using integer programming, is designed for Web-services. Hence, system resource management is merely assumed, which in the framework is the foundation for predicting and maintaining QoS at run-time.

Model transformation needs a formal way to specify both user and system QoS-requirements. Aagedals [10] work on Component-QML (CQML) and the UML profile for QoS [8] gives a formal lexical specification language suitable for both MDA-tools and run-time QoS analysis, and is used for specifying the QL-model and QL-mapping functions in the proposed service planning framework. In [22], Burt et al., explores how QoS requirements can impact decisions related to the transformation from platform-independent models in UML to platform-specific models in IDL. The idea of including QoS requirement in transformations is used as input to our approach using the service planning framework to perform QoS aware transformations. The approach does not address how QoS requirements can be integrated in a UML specification and how they could be resolved or refined in automatic model transformations. In [24], Schmidt et al. describes the CoSMIC framework, which describes an MDA-based development and run-time framework, which focuses on handling QoS policies in the run-time framework. It does not address the specification of QoS on a model level. The commercial MDA tools, such as ArcStyler and Codagen provide mechanism to support model driven processes including model transformations, but are not concerned about the integration of QoS in model abstraction.

Component technologies/architectures used in commercial products, like EJB [25], lack application programming interface (APIs) for adding QoS-management mechanisms. The component architecture OpenORB v2 [11] addresses this by introducing component frameworks (CF) as building blocks. Each CF has a set of policies and rules that provides QoS-support. The service planning framework supports CF, but the QoS meta-data is separated from the application and placed in a service plan. Another approach, [26], adds QoS-awareness to component based executions environments by extending the container with components and interface for QoS negotiation and adaptation. The service planning framework is based on a different philosophy, aiming for a technology neutral framework, which can be implemented in a range of executions environments. The middleware platforms OpenORB [11] and CARISMA [12] employ reflection for run-time reconfiguring. In OpenORB, applications can reconfigure the structure of the CFs, while CARISMA use reflection to add or change policies in the QoS-profile associated with the application. DynamicTAO adds reflection to CORBA, allowing inspection and reconfiguration of the ORB [27]. There are hooks for strategies that the ORB uses to implement middleware services, which may be replaced at run-time. These three alternative implementations require application code for QoS handling. The service planning framework does not require any application code, since the service plans capture the QoS-properties of the applications.

5. CONCLUSIONS AND FURTHER WORK

Business systems are now coupled tighter together, which has resulted in a need for QoS-awareness in both model-driven system development and the execution environment. Furthermore, with the introduction of SOC and Web-services the way one views enterprise application interfaces and component types has changed. In this paradigm everything is perceived as a service, and new services might be composed from existing services. The presented service planning framework extends existing modeling tools and execution environment with QoS-awareness.

The QoS-level provided by a distributed computing system is related to the availability of processing power, database access load, free memory space, networks technologies and traffic load. It is therefore important to allow the framework, at both build- and run-time, to work with alternative service compositions. Furthermore, due to changes in the execution environment, the framework must adapt the service composition to maintain the QoS-level. In the framework are concepts for specifying the QoS-properties of each service composition, enabling the framework to compare alternative service compositions based on their QoS-properties. When implemented in an execution environment user QoS-requirements and the system resource availability are inputs to the service planning framework The QoS-requirements sets the objective and system resource availability governs the achievable QoS-level. The composition with a QoS-level that meets the user QoS-requirements is chosen. To maintain the QoS-level the framework uses resource monitors to detect major changes in resource availability. If this happens it will result in a new search for a composition that meets the user QoS-requirements.

The framework can be applied for both external and internal service planning, i.e., for identifying and choosing a composition of Web-services or component types.

The video conference case, used to illustrate how one applies the framework in model-transformation and run-time planning, showed the importance of specifying formal QL-models and QL-prediction functions that enables both build and run-time service planners to make good decisions. How to express both the QL-models and QL-prediction functions needs to be studied more. Our current approach is limited to hard-coding, giving low flexibility and no reuse of models and functions. Design solutions for resource monitoring and interaction between peer service planners are also topics of further research.

6. REFERENCES

[1] Bowman, B., Debray, S. K., and Peterson, L. L. Reasoning about naming systems. *ACM Trans. Program. Lang. Syst., 15,* 5 (Nov. 1993), 795-825.

[2] Emmerich, W.: Distributed component technologies and their software engineering implications. In *Proceeding of the 24th international conference on software engineering*, ACM Press (2002)

[3] Papazoglou, M., Georgakopoulos, D.: Service-Oriented Computing. In *Communications of the ACM*, Volume 46, Number 10 (2003)

[4] F.Curbera et. al., Unraveling the Web Services: An introduction to SOAP, WSDL, and UDDI, IEEE Internet computing, vol.6, no.2 Mar./Apr. 2002.

[5] Dahl, O.-J., Myhrhaug. B and Nygaard, K., 1968, 1970, 1972, 1984: "SIMULA 67 Common Base Language", Norwegian Computing Center 1968 and later editions.

[6] Sun Micro Systems: *Java 2 platform enterprise edition specification*. v1.4 (2003)

[7] Soley, R.M, Frankel, D.S.,Mukerji, J.,Castain, E.H., Model Driven Architecture - The Architecture Of Choice For A Changing World, OMG 2001.

[8] UML™ Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms, Revised submission, August 18 2003, www.omg.org (members only)

[9] Zeng, L., Benatallah, B., Ngu, A., Dumas, M., Kalagnam, J., Chang, H.: QoS-Aware Middleware for Web Services Composition, *IEEE Transactions on software engineering*, Volume 30, Number 5 (2004) 311 – 327

[10] Aagedal, J. O.: *Quality of service support in development of distributed systems*. Ph. D. thesis (2001)

[11] Coulson, G., Blair, G., Clarke, M., Paralvantzas, N.: The design of a configurable and reconfigurable middleware platform: In *distributed computing*, Volume 15, Number 4, Springer-Verlag (2002) 109 – 126

[12] Capra, L., Emmerich W., Mascolo, C.: CARISMA: Context-aware reflective middleware system for mobile applications. In *IEEE transactions on software engineering*, Volume 29, Number 10 (2003) 929-945

[13] Walpole, J., Krasic, C., Liu, L., Maier, D., Pu, C., McNamee, D., Steere, D.: Quality of Service Semantics for Multimedia Database Systems. *Appears in Database Semantics: Semantic Issues in Multimedia Systems*, Kluwer Academic Publishers (1999)

[14] Staehli, R.: *Quality of Service Specification for Resource Management in Multimedia Systems*. In Ph.D. thesis (1996)

[15] Staehli, R., Eliassen, F.: *QuA: A QoS-aware component architecture*, Technical report Simula Research Laboratory (2002)

[16] Java Community Process, *UML profile for EJB*, JSR 26, http://www.jcp.org/jsr/detail/26.jsp.

[17] Object Management Group: *UML profile for schedulability, performance and time specification*. In ptc/2003-03-02, http://ww.omg.com (2003)

[18] Trading object service specification, 2000. http://www.omg.org/docs/formal/00-06-27.pdf

[19] UML™ 2.0, http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML

[20] UDDI version 3 specification, 2002. http://www.uddi.org/

[21] Staehli, R., Eliassen, F., Aagedal, J. O., Blair, G.: Quality of Service Semantics for component based systems. In *Proceedings for 2nd International workshop on Reflective and Adaptive Middleware Systems* (2003) 153 -157

[22] Burt, C.C., Bryant, B.R., Raje, R.R., Olson, A., Auguston, M.: Quality of Service Issues Related to Transforming Platform Independent Models to Platform Specific Models. In *Proceedings of EDOC 2002*, Lausanne, Switzerland (2002) 212-223

[23] MODA-TEL IST 2001-37785 deliverable 3.1, *"Model Driven Architecture Definitions and Methodology"*, R. Steinhau (Editor)

[24] Gokhale, A., Natarajan, B., Schmidt, D., Nechypurenko, A., Gray, J., Wang, N., Neema, S., Bapty, T. Parsons, J.: CoSMIC: An MDA Generative Tool for Distributed Real-time and Embedded Component Middleware and Applications. In *Proceedings of the ACM OOPSLA 2002 Workshop on Generative Techniques in the Context of the Model Driven Architecture*, (2002)

[25] Sun Microsystems: *Enterprise JavaBeans ™ Specification, Version:2.1*. (2002). http://java.sun.com/products/ejb

[26] Miguel A., Ruiz, J., Garcia, M.: QoA-Aware Component Frameworks. In *Proceedings of the 10th International Workshop on Quality of Service*. IEEE Communications Society (2002).

[27] Kon, F., Roman, M., Liu, P., Mao, J., Yamane, T., Magalhaes, L., Campbell, R.: Monitoring, Security, and Dynamic Configuration with the dynmicTAO Reflective ORB. In *Proceedings of Middleware 2000* (2000)

[28] Nahrstedt, K., Xu, D., Wichadakul, D, Li, B.: QoS-Aware Middleware for Ubiquitous Computing. *IEEE Communications Magazine*, Vol. 39, No. 11 (2001) 140 – 148

[29] Zeng, L., Benatallah, B., Ngu, A., Dumas, M., Kalagnanam, J. Chang, J.: QoS-aware middleware for Web Service Composition, *IEEE Transactions on software engineering*, Vol 30, No. 5. (2004)

Towards a Framework for Supporting the Negotiation between Global and Local Business Requirements

Paolo Traverso[1] Marco Pistore[2] Marco Roveri[1] Annapaola Marconi[1]
Raman Kazhamiakin[2] Pierluigi Lucchese[1] Paolo Busetta[1] Piergiorgio Bertoli[1]

[1]*ITC-irst, Via Sommarive 18, I-38050, Trento, Italy*
[2]*Department of Information and Communication Technology, University of Trento, Via Sommarive 14, I-38050, Trento, Italy*
{*traverso,roveri,marconi,lucchese,busetta,bertoli*}*@itc.it* {*pistore,raman*}*@dit.unitn.it*

ABSTRACT

The development of service oriented applications very often needs to address the problem of satisfying two conflicting kinds of business needs: *global business requirements*, i.e., the regulations that dictate the rules of engagement between different organizations, and *local business requirements*, i.e., the rules local to each involved partner which derive from its internal business needs. In this paper, we propose a development process where both global and local service requirements, as well as their behaviors, are incrementally agreed among partners and built through negotiation steps. The development process is supported by the explicit definition of both global and local requirements at different levels of abstraction. We express requirements in a language with a clear semantics, and which allows for explicit links to executable business processes, e.g., written in BPEL4WS. This development process opens up the possibility to adopt a variety of supporting techniques. In particular, automated verification is used to detect design or implementation problems. Automated synthesis of executable business processes allows for a speed up in the development process and reduces development effort. Finally, execution monitoring is able to detect run-time problems with respect to specified requirements.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques; D.2.1 [**Software Engineering**]: Requirements/Specifications; D.2.4 [**Software Engineering**]: Software/Program Verification; I.2.2 [**Artificial Intelligence**]: Automatic Programming; I.6.4 [**Simulation and Modeling**]: Model Validation and Analysis

General Terms

Design, Languages, Verification

Keywords

Service composition models and languages, Requirements for service-oriented processes

1. INTRODUCTION

In several application domains, service-oriented computing should provide a universal basis for the integration of business processes that are distributed across different entities, e.g., different organizations or companies. In these domains, different organizations must interact and cooperate according to global, shared requirements. At the same time, each organization has its own internal business needs, which are specific to the business it carries out. As a consequence, in these domains, two opposite and often conflicting kinds of business needs have to be taken into account. From one side, the *global business rules*, i.e., the regulations that dictate the rules of engagement between different organizations. From the other side, the *local business rules*, i.e., the rules local to each involved partner and deriving from its own internal business needs.

Several applications have this characteristic. This is the case, for instance, of several e-government applications involving different administrative offices or departments, where global rules derive from national or regional requirements and norms, while local rules derive from each office's responsibilities and internal organization. Another example are business coalitions or market-places, where different companies agree to obey to common market regulations, but still pursue their own distinct profit and interest.

In most of the cases, it is rather natural that global and local business rules have opposite goals and tend to conflict. For instance, a national law may require maximum transparency from a government office towards the citizen, e.g., the citizen should have the possibility to inquire at any time the status of any on-going procedure he is involved in. However, an administrative office may not like to be slowed down in its internal procedures with too many external interactions. A common market regulation may require that an offer evaluation is proposed to a customer after all partners' offers are available, while each vendor's need is to get to know as soon as possible whether the client will buy the product or not.

Dealing with the conflicts between global and local business rules, both valid and well motivated from the two different points of view, is what makes this kind of applications difficult to develop, and what makes them substantially different from traditional fully centralized applications, where an authority dictates the rules of the game, and fully distributed systems, where each actor has not to deal with general regulations and norms. As a consequence, the development process can hardly be carried out according to classical software development methodologies, which is not able to take into account both the global and the local rules and their natural conflicts. Moreover, software engineering tools that support the life-cycle of distributed business processes should be re-thought to support the development process that takes into account both the global and local business needs.

Within Astro, an Italian national project which aims at the study and application of service-oriented computing techniques, we are defining a novel development methodology and the supporting tools necessary to face the challenges outlined above. This paper describes this novel approach.

The central idea is that conflicting global and local business rules should be negotiated within the development process. More precisely, the proposed development process interleaves the phases of specification of both global and local rules with *phases of negotiation between global and local needs*. As a consequence, the *choreography*, i.e., the global view of how different partners interact, the *orchestration*, i.e., the description of how one partner interacts with (some of) the other partners, and the internal business process of each partner, are *incrementally built through negotiation steps*, thus emerging in a commonly agreed choreography and orchestration that, by obeying to global laws and norms, mediates among global and local needs.

To implement this development process based on negotiation, we propose a conceptual framework where the distinction between global and local business rules is explicit in the different phases of the development process, and at different levels of abstraction. We specify global and local business rules both at the level of *strategic requirements*, i.e., business goals and motivations, and at the level of *procedural requirements*, i.e., specifications on how a business should be carried out. The “transparency towards citizens” and the “internal administrative office efficiency” are two examples of, resp., global and local strategic requirements, while a process that does or does not allow for interaction at each step with the citizen is, resp., a global or local, procedural requirements. The proposed methodology also takes into account that not all the local rules of a given partner can be made visible to the other partners. For instance, a customer company may decide to keep confidential its internal business rules on how offers to customers are prepared, in order to keep a competitive advantage.

In this requirements driven development process, global and (external and internal) strategic and procedural requirements are stated with a precise notation and with a clear semantics, and are explicitly linked to the detail design and implementation of business process, e.g., written in standard business process modeling and execution languages, like (abstract or executable) BPEL4WS [1]. This opens up the possibility to provide tools that support the process based on negotiation during the development cycle: *verification tools* that detect specification, design or implementation problems, e.g., the fact the negotiation process leads to a choreography and/or orchestration that actually does not satisfy some global or local rule; *synthesis tools* that suggest solutions, like a business process design or implementation, to speed up the development process and reduce development effort; *monitoring tools*, i.e., tools that monitor the execution of a process to detect run-time problems w.r.t. requirements.

The described approach has been developed guided by a real application domain, investigated inside the Astro project. It consists of service-oriented applications for the public administration. In this domain, procedures involve several different administrative offices, which must follow the strict National and Regional laws and global policies concerning these procedures, but which should also preserve their own autonomy in order to deal with other tasks related to other procedures and to obey to its own internal requirements.

The paper is structured as follows. In Section 2, we introduce the application domain that will be used to explain our approach all along the paper. In Section 3, we describe the proposed development process supporting explicit negotiation phases. In Section 4, we explain how global and local rules can be described with a precise language for requirements specifications, while in Sections 5 and 6 we discuss how all of this opens up the possibility to construct tools that support the development process. We provide a discussion of related works and some concluding remarks in Section 7.

2. THE CASE: PUBLIC ENVIRONMENTAL AGENCY SYSTEM

The Environmental Protection Agency (EPA) is a local agency which deals with a wide range of environmental matters including protecting air, water and soil quality, managing waste, preventing or controlling pollution and promoting sustainable industry. To address these issues EPA has to deal with complex administrative procedures distributed among various actors (administrative offices as well as citizen and industries) and regulated by law: European, National, and Local norms contribute to specialize the same procedure adding new constraints, new actors and goals. Norms can be seen as a collection of goals and activities delegated to specific actors; moreover they specify constraints and obligations concerning for instance minimal and maximal durations of specific steps or of the overall procedure. The definition of a new procedure in the domain of Environmental Protection is a costly and time consuming task, that has to take into account constraints deriving both from norms and from the internal organizational structure of the actors involved in the procedure.

In this paper, we consider a specific licensing procedure for the establishment and operation of a Waste Disposal or Recycling Facility: A citizen or a company submits an application to obtain the license for its waste disposal or recycling facility (incinerator, private landfill,...); the local government, involving various agencies and experts, evaluates the proposal and authorizes it, if it complies with high standards dictated by norms. We will assume that the EPA wants to automate this procedure using web services interfaces, as they offer a platform independent approach for integrating applications.

According to the classical approach, EPA assigns the responsibility of mapping this procedure to a business analyst. Starting from specific regulations and norms (“D.C.I. 27 luglio 1984”; “L.R. 13 aprile 1995 n.59”; “D.Lgs. 5 febbraio 1997”) and interacting with the various actors involved, the business analyst models the procedure with the activity diagram depicted in Fig. 1. In this diagram, nested hexagon are used to describe the tasks and sub-tasks assigned to the different parties involved. Any citizen proposing to establish or operate a facility for solid wastes disposal has to apply for a certificate of designation; the application must be accompanied by all documents specified by the specific norms (e.g. an engineering design and operations report). The application is registered by the Protocol Office (PO) and then it is reviewed by the Waste Management Office (WMO) to determine whether the submitted documents are complete. If necessary, the WMO can ask the citizen to provide additional information or clarifications in order to complete the documentation. The validation of the documents

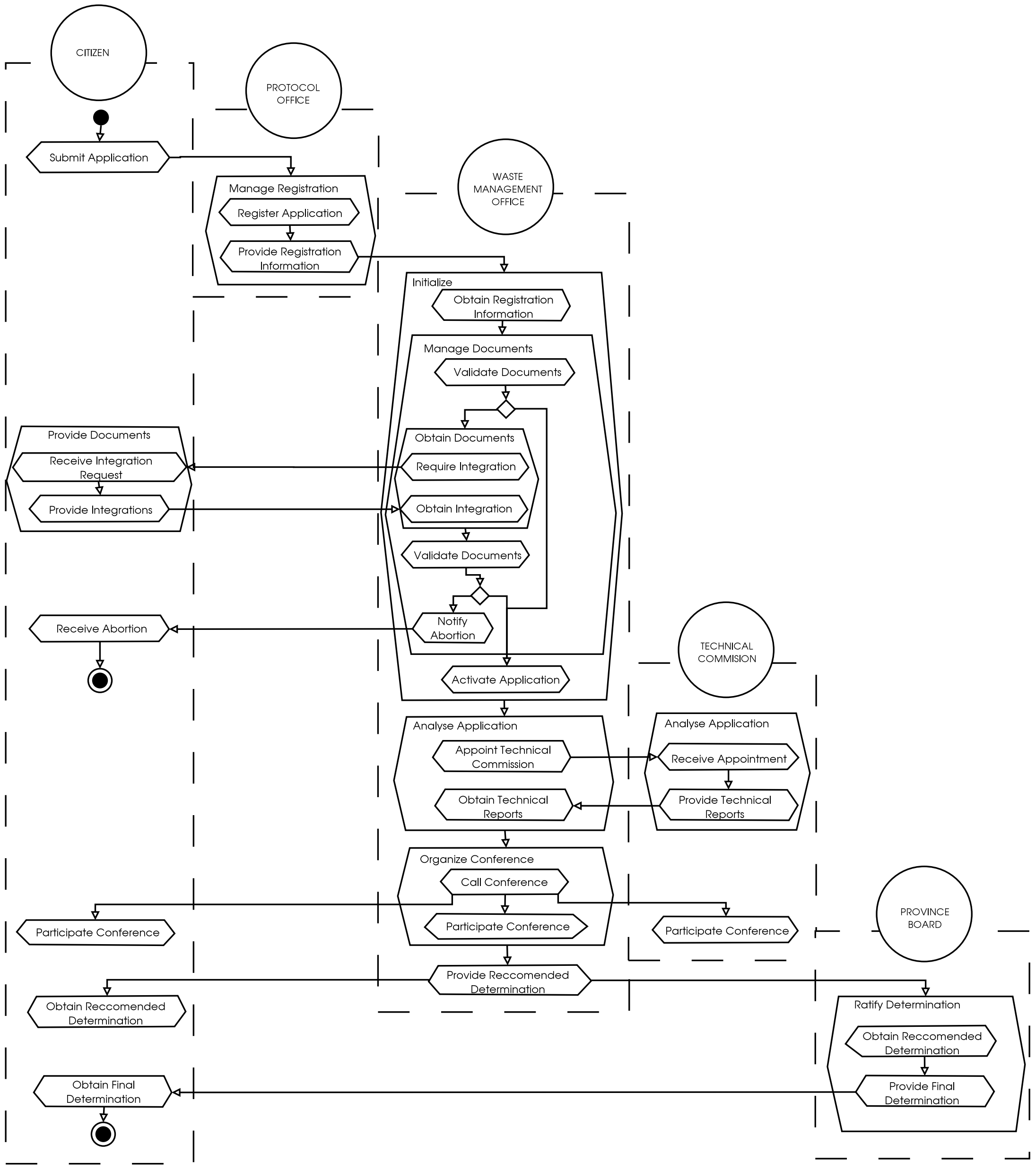


Figure 1: Activity Diagram for the Waste Facility License Procedure.

must complete within 30 days from the registration of the application. The WMO is then in charge of appointing the Technical Commission (TC), which is composed of various consultants and directors of public agencies (e.g. Sanitary Agency, Water Quality Control Agency, Soil Water and Plant Testing Laboratories, Environment Engineers,...). Each member of the TC has to produce a technical report and send it to the WMO which is responsible to set the Conference and to notify all the participants (TC members, citizen, WMO's responsible for the application,...). The aim of the Conference is to determine whether the facility complies with the norms, taking into account the submitted information and all the technical reports of the TC members. After the conclusion of the Conference the WMO will be in charge of producing the recommended determination and send it to the Province Board (PB) and to the citizen within 90 days from the Conference Day. The PB will evaluate the recommendations, draft the final determination and finally notify the citizen. Each application for a solid waste disposal site or facility should complete within 150 days from the PO registration. This global process defined by the business analyst is used as a blueprint for the design and implementation of the requested software components. In particular, starting from this global view, each actor involved defines (or adapt, if already existing) its internal processes and implements the web services necessary to carry out its part of the procedure.

The classical development approach outlined above is strongly based on a centralized, "authoritative" design that does not fit the requirements of distributed business processes. In particular, it does not take into account that the process developed by the analyst will have a high probability of being in conflict with the actor's internal requirements and constraints. A critical point is, for instance, the interaction between Citizen and WMO in order to complete the submitted documents. The norms regulating the procedure only require that the validation of the documents should terminate within 30 days. The Citizen would prefer to be able to submit new documentation incrementally within the 30 days, until the validation is successful. On the other hand, this iterative submission of documents would affect the efficiency of the WMO, since the scheduling of its work would depend on the Citizen. According to Fig. 1, the analyst has addressed this conflict by allowing the citizen to submit further information only once. However, from the diagram it is impossible to judge whether this is an acceptable compromise between these conflicting requirements. The negotiation-based development approach discussed in the next section addresses this kind of problems.

3. A DEVELOPMENT PROCESS BASED ON NEGOTIATION

The development process that we are designing is based on two principles: it is *requirements driven* and it is based on the *dichotomy between the choreography and orchestration* in the development of service oriented applications (see also Fig. 2). On the former principle we remark that a clear model of the conflicting requirements is necessary for being able to mediate among them. More precisely, we need to represent requirements at two different levels of abstraction: at a *strategic* level, for representing business goals and motivations, and at a *procedural* level, for describing how a business should be carried out. The activity diagram in Fig. 1 can be seen as a description of the procedural requirements, since it describes the way the procedure should be carried out. Strategic requirements include, e.g., the fact that the Citizen expects the WMO to be collaborative, while the WMO has the goal of reducing interactions. In Section 4 we will define suitable notations for representing them.

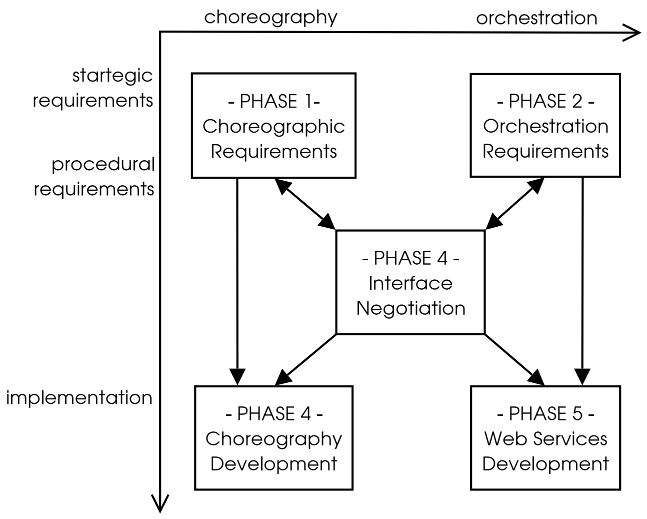

choreography
orchestration
startegic requirements
procedural requirements
implementation
- PHASE 1- Choreographic Requirements
- PHASE 2 - Orchestration Requirements
- PHASE 4 - Interface Negotiation
- PHASE 4 - Choreography Development
- PHASE 5 - Web Services Development

Figure 2: The Proposed Development Process.

The terms orchestration and choreography are often used to refer to the two key aspects of service oriented applications [13]. In *orchestration*, the application is considered from the perspective of one of the business parties. The focus is on the interaction that the party under consideration performs with internal and external web services in order to carry out its tasks inside the procedure. Orchestration is usually private to the business party, since it contains reserved information on the specific way a given process is carried out. *Choreography*, on the other hand, describes the interactions for a global, neutral perspective, in terms of valid conversations or protocols among the different parties. Choreography is usually public, since it defines the common rules for a valid composition of the distributed business processes in the business domain. In our process, we exploit the dichotomy between choreography and orchestration at all levels of the development. We will have both choreographic and orchestration descriptions of strategic requirements, of procedural requirements, and of the implementation based on web services.

The process we have been defining consists of five different phases (see Fig. 3). Taking into account the two principles just described, four phases correspond to the requirements analysis and to the implementation, done both from a choreographic and from an orchestration point of view. The fifth phase consists of the interface negotiation. This is the central phase of the whole process and plays the role of bridging between choreography and orchestration as well as between requirements and implementation. During this phase, the "choreographic" analyst responsible of the procedure and the "orchestration" analysts representing the different partners negotiate the design of the distributed application to be developed, mediating among conflicting goals. This negotiation phase terminates (and development starts) when an agreement has been reached on the services every partner should provide.

4. GLOBAL AND LOCAL REQUIREMENTS SPECIFICATION

Requirements play a fundamental role in the development process discussed in the previous section. Therefore, it is important to adopt flexible notations and methodologies for their specification. Activity diagrams like the one in Fig. 1 are fine for representing the procedural requirements, but they need to be completed with a description of the strategic requirements. We exploit the Tropos framework to this purpose. Tropos is a framework for the requirements-driven, agent-oriented development of software [2]. It is based on the premise that during requirements analysis it is important to understand and model the strategic aspects underlying the organizational setting within which the software system will eventually function. By understanding these strategic aspects one can better identify the motivations for the software system and the role that it will play inside the organizational setting. In previous works [5, 16] we have shown how Tropos can be adapted to represent the requirements of service-oriented applications.

Fig. 4 is an example of a Tropos diagram that provides high-level choreographic representation of the requirements of our case study. It describes the actors (circles) involved in the considered procedure with their strategic goals (the ovals attached to the actors). For instance in the diagram we have the Citizen that aims to obtain a waste facility license which is represented with the goal GetWasteFacilityLicence); the WasteManagementOffice that aims to handle with the several applications for getting a license (goal ManageApplication). The Tropos diagram also describes the *interactions* and *contracts* among the different parties. These interactions are represented at a strategic level by means of *dependencies* (the ovals linked to two different actors) that describe intent/offer matchings among actors. For instance the fact that the Citizen depends on the ProtocolOffice for the activation of the application to obtain a waste facility license is represented with the goal dependency ActivateApplicationManagement. Besides goals and dependencies, Tro-

Phase 1 – Choreographic requirements

Objective:	Define the requirements for the management of the procedure.
Responsible:	Business analyst in charge of the new procedure.
Input:	Description of the procedure (laws and regulations; discussions with the experts…). Definitions of the existing services that can be exploited in the procedure. Legacy systems that the procedure should reuse (e.g., existing centralized information systems).
Output:	Requirements specification document, covering a strategic (actors involved with their goals, responsibilities, mutual dependencies…) and a procedural (actors' tasks, control and data flows among actors and tasks…) description of the choreographic requirements for the new procedure.

Phase 2 – Actor's orchestration requirements

Objective:	Define the actor's requirements on the services it can provide to support the new procedure.
Responsible:	Business analyst of the specific actor.
Input:	Description of the roles and responsibilities of the actor inside the procedure (laws and regulations; discussions with the experts…). Internal requirements of the actor (i.e., business objectives, internal procedures and organization…). Definitions of actor's services and of other software that can be reused in the new procedure.
Output:	Requirements specification document, covering a strategic (actor's goals and responsibilities; assume/guarantee relations with external actors…) and a procedural (actor's tasks and task decompositions, internal business processes,…) description of the orchestration requirements of the specific actor inside the new procedure.

Phase 3 – Interface negotiation

Objective:	Define the interfaces of the web services provided by the different actors.
Responsible:	Board of the analysts responsible of Phases 1 and 2.
Input:	Requirements specification documents produced in Phases 1 and 2.
Output:	Definition of the web services provided by the different actors (and of the centralized systems) that permit the implementation of the procedure. The services are defined in terms of their interfaces (e.g., in WSDL), of the protocol for interacting with them (e.g., in business process specification languages like BPEL4WS), and, at the strategic level, of the tasks that the services are supposed to perform and the assumptions for their correct behavior.

Phase 4 – Development of the choreography

Objective:	Development of the centralized software systems (e.g., centralized information systems, wrappers for legacy systems…) necessary to support the new procedure.
Responsible:	Analyst / system architect responsible of the new procedure.
Input:	Definition of the choreographic requirements (Phase 1). Definition of the interfaces that the choreographic component should provide (Phase 3).
Output:	Detailed design and implementation of the choreographic system.

Phase 5 – Development of actor's web services

Objective:	Development of the web services of a specific actor.
Responsible:	Analyst / system architect responsible of the actor's software systems.
Input:	Definition of the orchestration requirements of the actor (Phase 2). Definition of the interfaces of the services that the actor should provide (Phase 3).
Output:	Detailed design and implementation of the actor's web services (or adaptation of the existing services and other software components).

Figure 3: The Proposed Development Process Phase-by-Phase.

pos permits to represent so called soft-goals and soft-dependencies (clouds). These represent non-functional requirements that will have an impact on how the procedure will be implemented, but whose achievement cannot be defined precisely in terms of clear cut properties (for instance, the appreciation is subjective, or the fulfillment of the requirement can occur only to a given extent). The goal of the Citizen of having a "transparent application management", or the dependency of having a "fair evaluation" from the Province Board are examples of these "soft" requirements.

It has to be noticed that in Fig. 4 and in Fig. 1 we have represented separately the strategic and procedural description of the *choreographic* requirements. However, these two diagrams are interconnected in the actual model of choreographic requirements. Indeed, each activity is linked with the strategic requirements that motivate its presence in the model and that define its expected behavior.

An example of a linked representation of strategic and procedural requirements, is provided in Fig. 5 from the local, "orchestration" point of view of the WasteManagementOffice. This diagram represents not only the global goals of the WMO already represented in Fig. 4 (shaded in the figure), but also its private goals representing the internal needs, requirements, and constraints of the WMO. The goals are organized in a tree structure that refines high-level goals into lower level goals, until they are operationalized into tasks. For instance the goal ManageApplication is refined in two sub-goals: Valid-

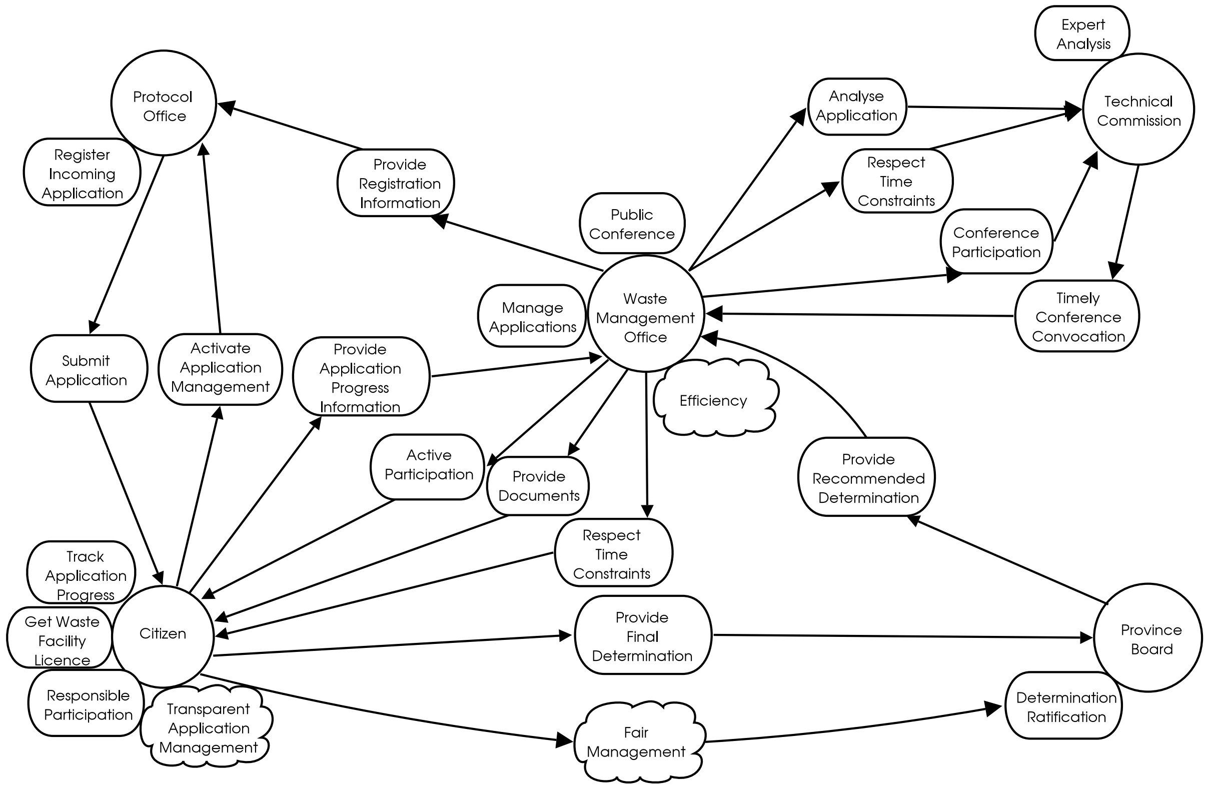

Expert Analysis
Protocol Office
Register Incoming Application
Provide Registration Information
Analyse Application
Technical Commission
Respect Time Constraints
Public Conference
Conference Participation
Manage Applications
Waste Management Office
Timely Conference Convocation
Submit Application
Activate Application Management
Provide Application Progress Information
Efficiency
Active Participation
Provide Documents
Provide Recommended Determination
Track Application Progress
Respect Time Constraints
Get Waste Facility Licence
Citizen
Provide Final Determination
Province Board
Responsible Participation
Transparent Application Management
Fair Management
Determination Ratification

Figure 4: The Choreographic Diagram of Strategic Requirements.

Documents and CompleteApplication. This decomposition is motivated by the requirements of having correct applications, which is captured by soft-goal Correctness. Indeed, in the diagram *contribution* links are used to represent the fact that two sub-goals contribute to the achievement of soft-goal Correctness. The goal ValidDocuments is further refined in the goals CompleteApllicationDocuments and ValidTechicalReports. These two goals are respectively operationalized with task ManageDocuments and ValidateTechicalReports. It has to be noticed that the latter task is not present in the choreography depicted in Fig. 1. Indeed this task is motivated by internal requirements of the WMO. In the diagram, two different kinds of links between goals and tasks are shown. Solid arrows are used to describe that some tasks have been obtained by the operationalization of certain goals, while dashed lines express the fact that the satisfaction of a certain requirement depends on a given task. One can see, for instance, that different tasks are responsible to guarantee the completion in time of the different phases of the procedure.

The Tropos notations discussed in this section are supported by a corresponding formal language, Formal Tropos [4], which allows for a precise definition of the requirements and of the activity diagrams and enables the usage of verification tools for detecting specification, design, or implementation problems. Formal Tropos permits to specify the valid behaviors and the relations among the different actors, dependencies, goals, and tasks that appear in a Tropos model. At the strategic level the Formal Tropos annotations specify properties like conditions on goal fulfillment, and assume/guarantee conditions on delegations. At the procedural level, they define pre- and post-conditions on tasks and sub-tasks. Even more important, Formal Tropos annotations allow to link together these two levels and the underlying implementation level. The key advantage of Formal Tropos with respect to other approaches is that it defines the dynamic aspects of a model and supports its formal verification already at the requirements level, without requiring an implementation of the specification, e.g., into BPEL4WS processes.

We conclude this section by remarking that, after the negotiation phase has been concluded, the refinement process of the requirements diagrams can further proceed transforming activity diagrams into executable code. In our framework we are adopting BPEL4WS [1] at the implementation level. BPEL4WS is quickly emerging as the language of choice for the description of process interactions. It provides core concepts for the definition of business process in an implementation-independent way, and allows both for the definition of internal business processes and for describing and publishing the external business protocol that defines the behavior of the interaction. Therefore, BPEL4WS permits to describe both the orchestration and the choreography of a business domain with an uniform set of concepts and notations. Most notably, BPEL4WS can be easily obtained by refining activity diagrams like the ones in Fig. 1 or in the bottom part of Fig. 5: see for instance [6] for a Model Driven approach to this refinement. Finally, as shown in [5, 16], links to the requirements can be maintained into the BPEL4WS code, so that requirements traceability is possible.

5. SUPPORTING THE PROCESS: VERIFICATION

The development process described before is accompanied by verification tools that support the different activities necessary to develop correct service oriented applications [5, 16]. These tools allow for verifying the correctness of a model at all levels of abstrac-

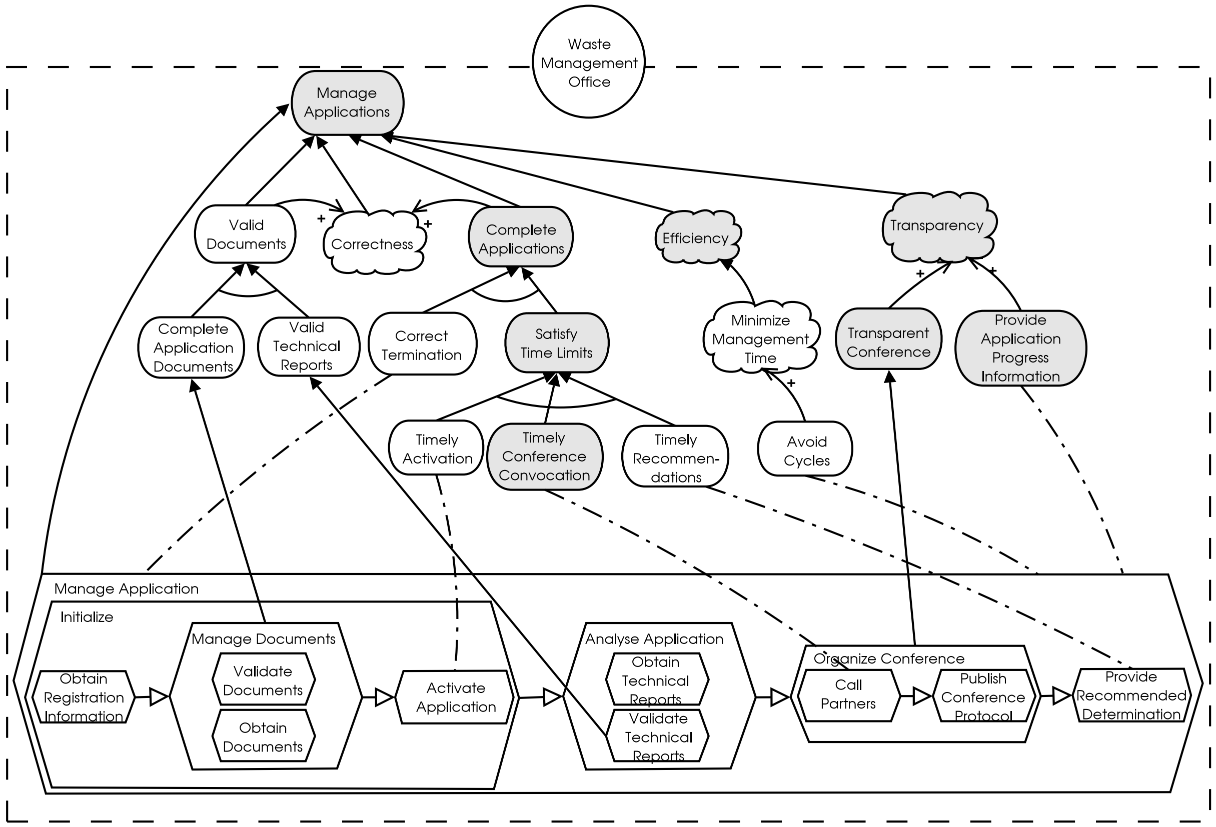


Figure 5: Strategic and Procedural Requirements of the WMO.

tions covered by our methodology. At the strategic level verification can be used to validate the requirements and to check their consistency. At the procedural level, verification can be used for proving that the processes are free of anomalies such as "deadlocks" (when a execution is "blocked" and no longer proceeds through the process) and "livelocks" (when an execution gets "stuck" in a never-ending loop), or to check the timing constraints on the different activities. At the implementation level, verification can point out incompatibilities and inconsistencies among the different web services that need to interact to carry out the procedure. Moreover, verification can be used to check the consistency among the different levels of a specification, that is, the procedural level of requirements should respect all constraints stipulated at the strategic requirements level, and the implemented web services should be a refinement of the activities defined at the procedural level. Finally, verification can be done both from a choreographic point of view, e.g., to check that the defined procedure respects all constraints imposed by the law, and from an orchestration point of view, e.g., to check that the services a party will offer are compatible with its own internal requirements and goal. For lack of space, we cannot give a comprehensive description of all applications of verification inside our process. We focus instead on some specific application scenarios.

A first usage of verification techniques is for validating choreographic requirements. While defining a global choreography, one should deal with partners interactions in terms of intent/offer matches as well as with the business rules common for all the participants of the business process. This makes the definition of this requirements model a complex and error-prone task. In order to catch misunderstandings and inconsistencies in this model one can verify it against set of properties that every execution of the system should satisfy (assertion properties) or some execution may satisfy (possibility properties). Querying the model allows one to check the correctness of the model with respect to the property or to check whether the model is not over-specified and some desirable behaviors are captured by the system. For instance, one property that the choreography should guarantee is that, if all actors carry out their own tasks, as described in the strategic requirements model, then the citizen will eventually get a (positive or negative) answer to the license request. However, a missing goal or dependency in the strategic requirements diagram may falsify this property. Suppose for instance that we remove dependency ActivateApplicationManagement between the Citizen and the PO from the requirements in Fig. 4. Then there is no guarantee that the Protocol Office will eventually forward our application to the WMO after having registered it, and the chain of activities leading to the answer to the citizen is broken. Indeed, if we exploit the verification techniques provided by Formal Tropos to verify that the GetWasteFacilityLicense goal of the Citizen will be eventually fulfilled, we will get a negative answer. Actually, the verification tool provides a counter-example scenario, showing that it is possible for the PO to fulfill all its goals without having to forward to the WMO the citizen's application.

At a lower level of abstraction, verification can be used to detect anomalies like deadlocks in the activity diagrams defining the interactions among parties. For instance, let us assume that, within the negotiation process, we modify the definition of the ManageDocuments activity in Fig. 1 as described in Fig. 6. The intuition is that we want to model a WMO that keeps interacting with the citizen in

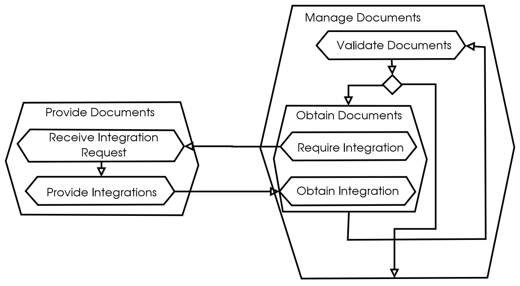
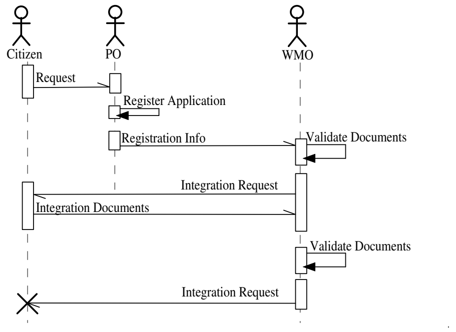


Figure 6: Modified Citizen - WMO interaction.

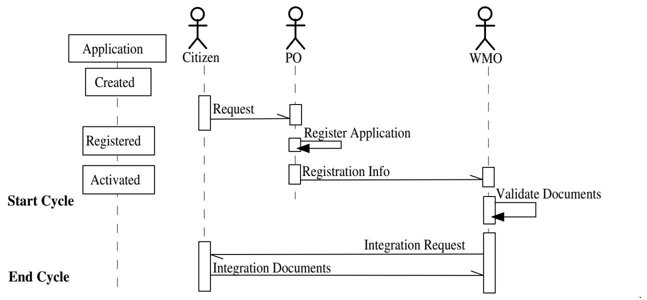
an iterative, cyclic way until a complete documentation is obtained. If this modification is not reflected into the orchestration activities of the Citizen, a deadlock occurs. Indeed, if the documents provided are not correct also after a first integration, the WMO will ask for further documents. However, according to Fig. 1, after a first integration the Citizen expects either a conference announcement or an abortion of the procedure, so he is not able to provide further documents. The verification techniques we are providing can be used for finding such inconsistencies. For instance, if the analyst designing the services of the Citizen verifies his internal process against the modified choreography described above, the inconsistency is detected and the following scenario leading to the deadlock condition is reported as a witness:



A last example of verification consists in checking if the choreographic process model is compatible with the local needs and expectations of a specific actor. Let us assume that the choreographic process adopted permits a cyclic interaction between the WMO and the Citizen in order to obtain integration documents, as in Fig. 6. Then the verification tool shows that the internal goal CorrectTermination of the WMO may be violated. Indeed, the formal specification of the goal is that every application submitted to the WMO should terminate with a recommendation or should be eventually aborted by the WMO. This property is violated if the choreography allows for cyclic interactions with the Citizen, and the following example of goal violation is reported by the verification tool:

6. SUPPORTING THE PROCESS: SYNTHESIS AND MONITORING

In this section we comment on how we can exploit program synthesis techniques to automate the development of web services within our reference process. These techniques come to help after the design and the negotiation of the web services has been done and every participant has to implement his own services. The scenario we are interested in is when the participant already has services (or



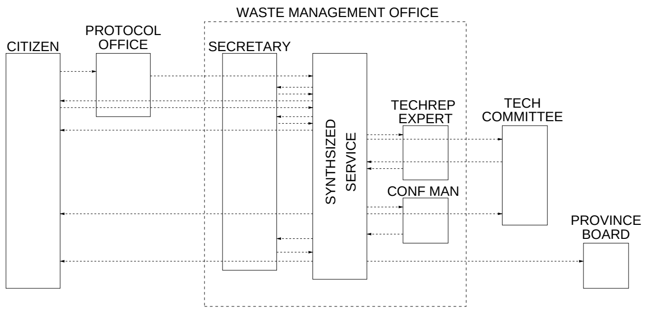


Figure 7: Synthesis of the WMO Service.

other software components) available that can be exploited to carry out his activities, but these services have to be adapted and composed in a way suitable to the new procedure ad hand. In our case study, the WMO already has internal services available for managing the standard tasks occurring in the different procedures the office is involved in. These services represent in some sense the 'back-office' of the WMO (see Fig. 7). In our case, the back-office consists of a Secretary service, of a Technical Report Expert and of a Conference Management. The secretary is in charge of evaluating whether the documents provided by the user conform to the requirements, to incrementally file the evolution of the request, and to extract relevant data that have to be communicated to the other parties involved. The technical report expert is in charge to communicate the relevant data to the technical committee, and collect and interpret his responses. The conference management is in charge of organizing the meetings between the participants involved. To participate to the new procedure, the WMO has hence to implement one more service that interacts with the other internal "back-office" services and with the external services of the other participants.

Automating the generation of this new service can be seen as a particular instance of automated generation of web services. By automated composition [14, 20] we mean the task of generating automatically, given a set of available web services, a new web service that achieves a given goal, i.e., that satisfies a given requirement, by interacting with the available web services. Different techniques have been proposed so far which address this problem. In our framework, we exploit the automated task planning techniques described in [14, 20]. According to the approach of [14, 20] (see Fig. 8), we take as our starting point the BPEL4WS specification of the existing internal and external services ($W_1, \ldots, W_n$). In our case, these descriptions are either already available to the WMO (for the internal services) or they are an outcome of the negotiation phase (for the external services). We encode each of the BPEL4WS specification in a state transition system ($\Sigma_{W_1}, \ldots, \Sigma_{W_n}$ in Fig. 8), which provides a sort of operational semantics to the BPEL4WS model. Each of them describes the corresponding web service as a

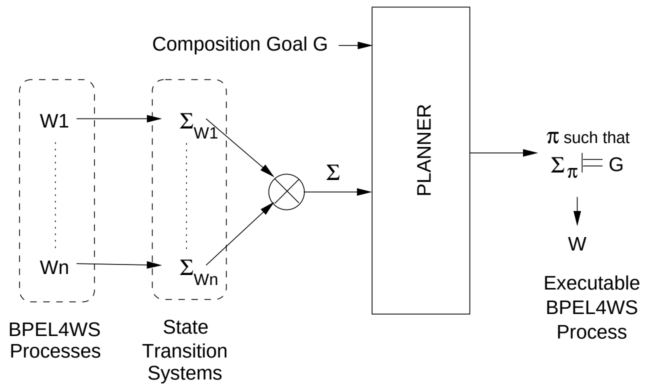


Figure 8: Automated Composition.

state-based dynamic system, that can evolve, i.e., change state, and that can be partially controlled and observed by external agents. In this way, it describes a protocol that defines how external agents can interact with the service. From the point of view of the new composed service that has to be generated, say W, the state transition systems $\Sigma_{W_1}, \ldots, \Sigma_{W_n}$ constitute the environment in which W has to operate, by receiving and sending service requests. They constitute what, in planning literature, is called a planning domain, i.e., the domain where the planner has to plan for a goal. In our case, the planning domain is a state transition system Σ that combines $\Sigma_{W_1}, \ldots, \Sigma_{W_n}$. Formally, this combination is a parallel composition, which allows the n services to evolve independently and concurrently. Σ represents therefore all the possible behaviors, evolutions of the planning domain, without any control performed by the service that will be generated, i.e., W. The composition goal G (see Fig. 8) imposes some requirements on the desired behavior of the planning domain. In our case, the goal can be obtained from the "orchestration" requirements model of the WMO. Given Σ and G, the planner generates a plan π that controls the planning domain, i.e., interacts with the external services $W_1, \ldots, W_n$ in a specific way such that the evolutions satisfy the goal G. The plan π encodes the new service W that has to be generated, which dynamically receives and sends invocations from/to the external services $W_1, \ldots, W_n$, observes their behaviors, and behaves depending on responses received from the external services. The plan π is encoded as an automaton and can hence contain complex control constructs, like tests over observations, conditionals, loops, etc. As a final step, we can translate π into process executable languages, like BPEL4WS.

Though still preliminary, the experiments reported in [14, 20], show that the automated synthesis approach described above can deal with cases that are far from trivial. Moreover, an interesting possibility offered by the composition approach described in Fig. 8 is that of obtaining *monitors*, i.e., software components that are able to observe the messages exchanged with (internal or external) services and to report whether they are violating the BPEL4WS protocol that they are supposed to implement. Indeed, we can exploit to this purpose the finite state machines $\Sigma_{W_1}, \ldots, \Sigma_{W_n}$, that capture the operational semantics of the corresponding BPEL4WS specifications.

7. CONCLUSIONS AND RELATED WORK

In this paper we propose a development process where global and local requirements are incrementally defined within a negotiation process. Requirements are described in a language with a clear semantics, which allows us to define precise links between business requirements and (executable) business processes. This opens up to the construction of tools for the analysis of requirements, the verification of business processes, as well as their synthesis and monitoring. The proposed approach has been inspired by and experimented with a real application we are developing in the public administration field. We see this work as a first step towards the construction of techniques and tools that support the development of distributed services by reducing development time, efforts, and errors.

The Model Driven Architecture [3], backed by OMG specifications such as UML 2.0 [18, 17] , aims to separate business logic from the details of platforms, programming languages and middleware. Developers create platform-independent models (PIMs), which can be semiautomatically transformed to platform-dependent models (PSMs). We share with this approach the need for high level specifications of services; more specifically, in terms of "Model Driven Service Composition" [11], we share the idea that service requirements should be analyzed in a systematic way, and the idea to describe business rules as precise statements. However, none of the previous approaches is based on the idea of incremental definition of the business rules that come out of a negotiation between global and local goals. In our proposal, requirements are structured, analyzed, and negotiated according to a clear distinction between internal business needs for a single business process, dependencies of objectives among different partners, and business rules that are common to a community of services. The development process described in other works, see, e.g., [11, 12], focuses on an important but orthogonal issue, i.e., how high level requirements (e.g., expressed in UML and OCL [11]) for a single process can be classified for service composition, and how they can be refined into executable processes. Some of the model driven approaches advocate for the use of verification techniques, e.g., based on Petri nets [15], or model checking [7, 8]. However, in these approaches, the problem of verifying local versus global rules is not addressed.

In [21] a formalism is proposed, based on Petri nets, which allows for verifying that local implementations of workflows do not create anomalies over organizational borders. However, the considered development process is purely top-down, from community requirements to the local implementation of workflows that have to satisfy the global requirements. There is no global-local requirements negotiation in [21], and thus the problem of verification of local versus global requirements is not addressed.

Different automated planning approaches have been proposed for the composition of web services [22, 10], for the interactive composition of information gathering services [19], and for providing viable plans satisfying specific queries of the user [9]. Within the development process that we propose, we use instead automated planning techniques to generate automatically executable business processes from high level requirements, and to generate automatically at design time monitors that can detect problems at run-time.

Acknowledgments

This work has been supported in part by the FIRB-MIUR project RBNE0195K5 "Astro". The authors want to thank all members of the Astro project for their collaboration and their feedback.

8. REFERENCES

[1] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, F. Leymann J. Klein, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. *Business Process Execution Language For Web Services, Version 1.1*, 2003.

[2] J. Castro, M. Kolp, and J. Mylopoulos. Towards requirements-driven information systems engineering: the Tropos project. *Information Systems*, 27(6):365–389, September 2002.

[3] D.S. Frankel. *Model Driven Architecture: Applying MDA to Enterprise Computing*. John Wiley and Sons, 2003.

[4] A. Fuxman, L. Liu, J. Mylopoulos, M. Pistore, M. Roveri, and P. Traverso. Specifying and analyzing early requirements in Tropos. *Requirements Engineering*, 2004. To appear.

[5] R. Kazhamiakin, M. Pistore, and M. Roveri. A framework for integrating business processes and business requirements. In *Proc. 8th Int. IEEE Enterprise Distributed Object Computing Conference (EDOC'04)*, 2004. To appear.

[6] J. Koehler, R. Hauser, S. Kapoor, F. Y. Wu, and S. Kumaran. A model-driven transformation method. In *Proceedings of the Seventh International Enterprise Distributed Object Computing Conference (EDOC'03)*, pages 186–197, Brisbane, Queensland, Australia, September 2003. IEEE Computer Society.

[7] J. Koehler, R. Hauser, S. Kapoor, F.Y. Wu, and S. Kumaran. A Model-Driven Transformation Method. In *EDOC 2003*, pages 186–197. IEEE Press, 2003.

[8] J. Koehler, G. Tirenni, and S. Kumaran. From Business Process Model to Consistent Implementation: A Case for Formal Verification. In *EDOC 2002*, pages 96–106, 2002.

[9] A. Lazovik, M. Aiello, and Papazoglou M. Planning and Monitoring the Execution of Web Service Requests. In *Proc. of the 1st International Conference on Service-Oriented Computing (ICSOC'03)*, 2003.

[10] S. McIlraith and S. Son. Adapting Golog for composition of semantic web Services. In *Proc. 8th International Conference on Principles of Knowledge Representation and Reasoning*, 2002.

[11] B. Orriens, J. Yang, and M. Papazoglou. Model Driven Service Composition. In *Proc. of the 1st International Conference on Service-Oriented Computing (ICSOC'03)*, 2003.

[12] M. Papazoglou and J. Yang. Design Methodology for Web Services and Business Processes. In *Proc. of the Technologies for E-Services Third International Workshop (TES'02)*, pages 54–64, 2002. Lecture Notes on Computer Science, Springer-Verlag, Berlin, Germany.

[13] C. Peltz. Web Services Orchestration and Choreography. *Web Services Journal*, July 2003.

[14] M. Pistore, F. Barbon, P. Bertoli, D. Shaparau, and P. Traverso. Planning and monitoring web service composition. In *Proc. 11th Int. Conf. on Artificial Intelligence: Methodology, Systems, Architectures*, 2004. To appear.

[15] D. Quartel, M. van Sinderen, and L. Ferrera Pires. Service Creation: a model based approach. In *Proc. of the 7th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'99)*, 1999.

[16] M. Roveri R. Kazhamiakin, M. Pistore. Formal Verification of requirements using Spin: A Case Study on Web Services. In *Proceedings of the 2nd International Conference on Software Engineering and Fomal Methods (SEFM'04)*, Beijing, China, 2004. IEEE Computer Society.

[17] P. Rivett and OMG Group. Unified Modeling Language: Infrastructure - version 2.0, 2003.

[18] B. Selic and OMG Group. Unified Modeling Language: Superstructure - version 2.0, 2003.

[19] S. Thakkar, C. Knoblock, and J.L. Ambite. A View Integration Approach to Dynamic Composition of Web Services. In *Proceedings of ICAPS'03 Workshop on Planning for Web Services*, Trento, Italy, June 2003.

[20] P. Traverso and M. Pistore. Automatic composition of semantic web services into executable processes. In *Proceedings of 3rd International Semantic Web Conference (ISWC2004)*, Lecture Notes Computer Science, Hiroshima, Japan, 2004. Springer Verlag.

[21] W.M.P. van der Aalst. Inheritance of Interorganizational Workflos: How to agree without loosing control? *Information Technology and Management Journal*, 2(3):195–231, 2002.

[22] D. Wu, B. Parsia, E. Sirin, J. Hendler, and D. Nau. Automating DAML-S Web Services Composition using SHOP2. In *Proceedings of the Second International Semantic Web Conference (ISWC2003)*, 2003.

Interface inheritance for object-oriented service composition based on model driven configuration

Vincenzo D'Andrea
DIT, Univ. of Trento
Via Sommarive, 14
38100 Trento
Italy
dandrea@dit.unitn.it

Ioannis Fikouras
BIBA, Univ. of Bremen
Hochschulring 20
28359 Bremen
Germany
fks@biba.uni-bremen.de

Marco Aiello
DIT, Univ. of Trento
Via Sommarive, 14
38100 Trento
Italy
aiellom@dit.unitn.it

ABSTRACT

In today eCommerce environments, customers have to deal with a wide variety of alternatives, both in terms of service offerings as well as service providers. They risk to be overwhelmed by the complexity of alternatives, thus reducing the usefulness of the experience and consequently the likelihood of transactions. There is an increasing need for new ways to reduce the perceived complexity. Service-oriented computing can help the user cope with this problem. With services, interfaces no longer hide units of code, but provide access to complex functionality equivalent to that of entire conventional applications.

We introduce a methodology for extended service composition derived from model-driven configuration and object-oriented systems. By focusing on the concept of interfaces, and applying it to the object-oriented concept of inheritance, we propose an innovative approach to composition that takes into account how the *composed* services can be recognized or accessed via the *composing* service. In order to set the stage, we discuss the similarities between Service Oriented Computing, Object-Oriented Configuration and Object-Orientation. In addition, we provide an overview of knowledge-based systems, described as software systems built by capturing the knowledge used by experts, and more specifically object oriented configuration for implementing service composition.

Categories and Subject Descriptors

H.1 [**Information Systems**]: Models and Principles; D.1.5 [**Software**]: Programming techniques—*Object-oriented Programming*

General Terms

Web services, Object-oriented programming, Model driven configuration

1. INTRODUCTION

"It is the customer who determines what a business is" [13] by attempting to address specific needs and express his personality through custom-made products and services [26]. Customers thus drive vendors to strive for product palettes with an ever increasing number of variants. Consequently the pursuit of differentiation through variety leads to unique products and services [16, 19]. This strategy is known as "mass customization". Mass customization is defined as "when the same large number of customers can be reached as in mass markets of the industrial economy, and simultaneously they can be treated individually as in the customized markets of pre-industrial economies" [12]. According to [31] the objective of mass customization is "to deliver goods and services that meet individual customers needs with near mass production efficiency". Online transactions and specifically eCommerce environments differ greatly from conventional commercial transactions. Online transactions achieve greater execution speeds and can bridge greater distances than traditional commerce. Furthermore purely digital products (i.e., information services or digitized media) can be discovered, adapted, evaluated, purchased, paid for and delivered by a single service platform within a very short timeframe at any time of day or place on earth [32]. Moreover such platforms compared to conventional sales facilities (i.e., brick and mortar stores) are quick and cheap to implement as well as adapt to new requirements even in not previously predetermined ways [32]. This allows for the rapid and inexpensive deployment of on-line stores offering advanced functionality (such as rearranging the product palette for individual customers) impossible to implement in brick-and-mortar facilities.

On the other hand, customers in an eCommerce environment are faced with more information, resulting from a wider variety of alternatives both in terms of service offerings as well as service providers. However the processing of this information occurs based on the same knowledge and information processing capacity available to the customer as in conventional shopping scenarios [32]. These constraints, unaffected by new technologies, result in a significant drawback to high variety strategies. A customer overwhelmed by the amount of available products or frustrated by their complexity is less likely to complete the transaction and purchase the product, and more likely to delay the decision or leave the shop altogether [18]. This behaviour illustrates the need for new ways for retailers to reduce the perceived complexity of their products. Advanced functionalities are designed to help the user cope with a large amount and at the same time a significant complexity of product data. The advance functionality necessary to accomplish the vision of mass customization may be offered by service composition functionality implemented in a service-oriented infrastructure.

Service oriented computing (SOC) is a new computing paradigm in which complex systems are built on the basis of basic distributed autonomous services by abstracting on the actual implementation and location of the various services [24]. This paradigm allows for a high distribution of the workload, for the building of complex system yet dynamically and easily scalable. Following the "Service Oriented Computing Manifesto" [25], SOC is more formally defined in terms of services, that is:

> Services are autonomous platform-independent computational elements that can be described, published, discovered, orchestrated and programmed using XML artifacts for the purpose of developing massively distributed interoperable applications.

The best-known example of service-oriented technology is that based on web services. In [10], web service are described as

> a networked application that is, able to interact using standard application-to-application Web protocols over well defined interfaces, and which is described using a standard functional description language.

In the SOC paradigm the emphasis shifts from the engineering of appropriate isolated applications towards the integration, orchestration and choreography of a set of independent services over a network. Typical distributed systems properties [8] become of paramount importance in this setting, most notably: heterogeneity, openness, security, scalability, failure handling, concurrency, transparency. Furthermore, in the SOC model no fixed synchronous bindings are established, but rather the computational elements follow the *find-bind-use* model.

If the scene is that of a web of autonomous computational elements that offer simple services exposing their interfaces, then the challenge is that of creating massively distributed applications offering added value by taking advantage of the basic services. In other words, service composition is the cornerstone for the success of the SOC vision.

Various approaches to service composition have been proposed in the literature. On one extreme are those who consider composition as a run-time process in which services are composed on the fly, e.g., [20]. To achieve this, semantic annotation of services going beyond a simple operational interface is mandatory. Efforts involving semantic web technology are blooming, most notable is the semantic web service initiative (`www.swsi.org`), but others based on temporized automata have also been proposed [4]. On the other extreme, many approaches consider composition as an engineering process that starting from user requirements, data or knowledge models arrives at a service composition satisfying the requirements. Examples of this approach are [7, 23].

In [14], we have shown how Model Driven Configuration theory can be exploited for service composition and orchestration, in [11] we have shown the analogies relating object-oriented programming and service-oriented design. In this paper, we propose a methodology for extended service composition derived from model-driven configuration and object-oriented systems, having the notion of service as the central building block. By focusing on the concept of interfaces and applying it to the object-oriented concept of inheritance, we propose an innovative approach to composition that takes into account how the *composed* services can be recognized or accessed via the *composing* service. We propose a classification of service composition, derived from the concepts of inheritance, interface inheritance, and object composition. For instance, from the notion of object composition we derive the definition of *Opaque Composition*, that is, a service is composed by other services without informing the external world of the details of the composing services.

The paper is organized as follows. In Section 2 we provide an overview of knowledge based systems. The paper then proceeds focusing on the use of knowledge-based construction systems, specifically object oriented configuration for implementing service composition. In Section 3, we present a discussion of the similarities between Service Oriented Computing, Object-Oriented Configuration and Object-Orientation, in order to bridge the gap between model driven configuration and services. Section 4 presents the main results of the paper, that is, a methodology for the composition of services based on object-oriented configuration. Concluding remarks and open issues are summarized in Section 5.

2. KNOWLEDGE-BASED SYSTEMS

We focus on the use of knowledge-based construction systems, specifically model-driven variant configuration for implementing service composition. The following section gives thus a broad overview of knowledge-based systems.

Knowledge-based systems are defined in [1] as:

> computer programs which (a) use knowledge and inference procedures (b) to solve problems which, if addressed by a human, would be regarded as difficult enough to require significant expertise.

For the purposes of this paper we use the following definition of *software systems built by capturing the knowledge used by experts and structuring it according to a specific method, in order to solve problems requiring application domain specific knowledge.* Such knowledge stored in a knowledge-base can be organized according to a number of different methods depending on the underlying concept for storing and managing knowledge. Methods in use include rules-based systems using lists of rules for describing dependencies and conditions, case-based systems that use libraries of predefined descriptions of past cases, and finally object oriented or model-driven systems that store knowledge in an object hierarchy with the help of a domain specific data model [29]. Furthermore knowledge-based systems are split into three broad categories Diagnosis, Simulation and Construction, according to the type of problem they attempt to solve [27, 28]. In our work, we focus on the category of Construction and, in particular, on Configuration problems.

The goal of Construction is the creation of a new solution out of a set of existing components. Construction problems include the configuration of products, processes, resources or services. Configuration is the design of a system through identification, parameterization and combination of instantiations of appropriate components types out of a predefined component set [17]. Configuration focusing on the modification of existing constructions is termed Variant Configuration.

Variant Configuration [29] is a process were complex products are composed out of elementary components. A configurator in this sense is a knowledge-based system implementing such process, based on predefined goals as well as domain specific knowledge. Design goals can be constraints, functional requirements, predetermined components or various quality criteria [21]. Such systems do not follow a single predefined method, but rather a strategy based on a series of small steps, each step representing a certain aspect or assumption leading to the configuration of the composite service. Configuration is therefore considered as the solution to a single exercise and not the solution to a whole problem or problem class that has first to be methodically analyzed. This implies the following, see Figure 1:

- The set of all possible solutions is finite.
- The solution sought is not innovative, but rather is a subset of the available parts.
- The configuration problem is known and well defined.

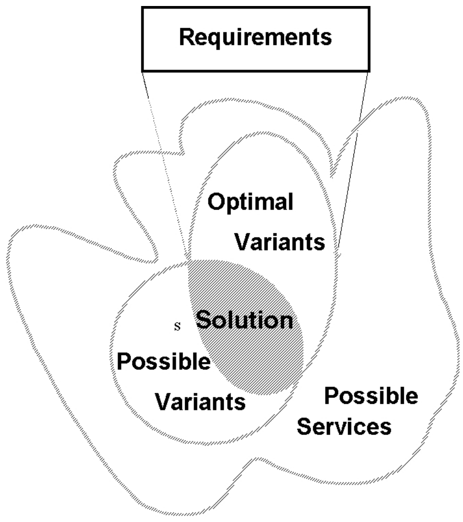


Figure 1: Variant Configuration solution-space.

Configuration as a knowledge-based system requires a knowledge-base as the source of its domain specific knowledge. The structure of this knowledge-base determines to a large degree the configuration process itself. Currently three major types of variant configuration are defined: (i) rules based configuration, (ii) case-based configuration, and (iii) model driven or object oriented configuration.

Object-oriented Variant Configuration is based on the concept of iterative composition of the final product out of a set of elementary components that have been previously organized according to a product data model into a structure, known as the object hierarchy that contains all knowledge related to the product in question. The relationships between components and how they fit together are described with the help of constraints.

Constraints are constructs connecting two unknown or variable components and their respective attributes which have predefined values (taken from a specific knowledge domain). The constraint defines the values the variables are allowed to have, but also connects variables, and more importantly, defines the relationship between the two values [30]. In other words, constraints contain general rules that can be applied to make sure that specific components are put together in a correct fashion without having to specify any component-related rules or calculations [30]. The constraint satisfaction problem is defined as follows [2]:

- There is a finite set of variables $X = \{x_1, \ldots, x_n\}$.
- For each variable x_i, there exists a finite set D_i of possible values (its domain).
- There is also a set of constraints, which restrict the possible values that these variables are allowed to take at the same time.

The object hierarchy contains all relevant objects and the relationships between them in an “is-a” relationship that defines types of objects, object classes and subclasses, and their properties. The configuration process creates objects on the basis of this information according to the products being configured. In one specific hierarchy (as depicted in Figure 2 for the configuration of automobiles), classes for specific car types (i.e., coupè, minivan, etc.) are connected by “is-a” relationships to the main “car” class. This hierarchy also allows the breakdown of a product into components with the help of further “has-parts” relationships. These “has-parts” relationships are the basis for the decision-making process employed to create new configurations. An example of such a relationship would be the relationship between a chassis and a wheel. A chassis can be connected to up to four wheels in a passenger car, but the wheels are represented only once, with appropriate cardinality.

The greatest hurdle to be resolved when creating new configurations is the fact that the software is required to make decisions that are not based on available information. Such an action can possibly lead to a dysfunctional composition or simply to a combination that does not conform to user requirements. In this case all related configuration steps have to be undone (backtracking) in order to return to a valid state. The longer it takes for the configuration to detect that a mistake has been made, the more difficult it is to correct the error in question [21]. The configuration process itself is composed of three phases [9]:

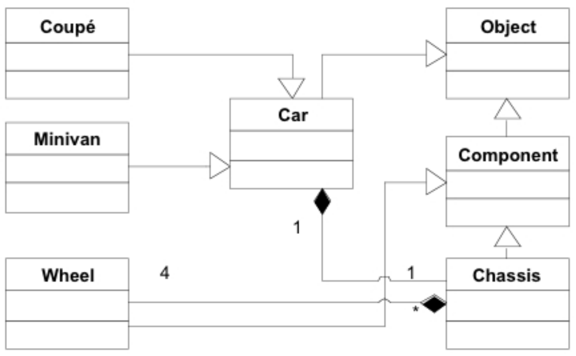


Figure 2: Automotive object hierarchy.

- Analysis of the product in order to define possible actions.
- Specification of further configuration actions.
- Execution of specified actions.

These actions are:

- Disassembly of the product into its components. This is meant to reduce the complexity of the problem and create a large number of smaller objectives in the manner of conventional top-down specification.
- Assembly of components, integration and aggregation. This step creates a product out of its components in a bottom-up manner.
- Creation of specialized objects. Object classes are specialized through the definition of subclasses.
- Parameterize objects. Define attributes and parameters for the specified objects that can be used for the application of constraints or other configuration mechanisms.

3. MODEL DRIVEN CONFIGURATION AND OBJECT ORIENTATION

A service composition engine based on object-oriented configuration implemented by project NOMAD [22] employs the following data model for composition of services. It divides services conceptually into two categories, Elementary Services and Composite Services, cf. Figure 3. Elementary Services represent a specific instantiation of a service and contain all data needed to describe it. Composite Services act as templates designed to provide the default knowledge required to produce a specific composition and consist of groups of Components derived individually from Elementary Services. Interfaces can be defined between Elementary Services, Composite Services, Service Categories and Service Providers. For a detailed discussion of the NOMAD service composition data model the reader is referred to [15].

The relationship between interfaces and elementary services matched by the filters contained in an interface resembles

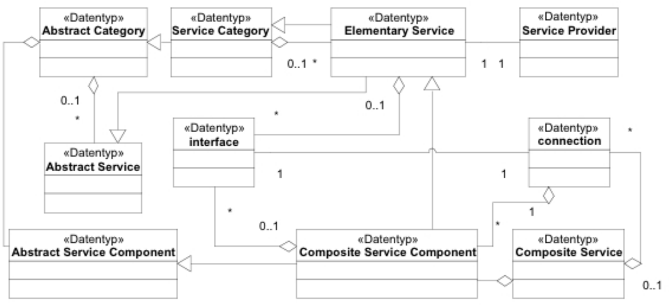

«Datentyp» Abstract Category
«Datentyp» Service Category
«Datentyp» Elementary Service
«Datentyp» Service Provider
0..1 *
1 1
0..1
*
0..1
*
«Datentyp» Abstract Service
«Datentyp» interface
«Datentyp» connection
*
1
1
1
*
0..1
*
«Datentyp» Abstract Service Component
«Datentyp» Composite Service Component
«Datentyp» Composite Service
0..1

Figure 3: Object hierarchy for composition of services.

the one between plugs and sockets, whereby interfaces as sockets match multiple plugs. Henceforth, connections to Elementary Components that have a direct reference to an interface via its unique identifier will be referred as *sockets* and components that are matched by a socket will be referred to as *plugs*. An interface object is not restricted in its scope to use by only one pair of services, but rather implements a generic rule (constraint) that can be used by multiple components for describing their interfaces. For a detailed discussion of the NOMAD service composition engine the reader is referred to [14].

One of the common metaphors used in textbooks on Object-Oriented programming (OOP) is to view objects in terms of the services they provide, describing them in "service oriented" terms (see for instance [5]). Building on abstraction and encapsulation, the key idea is to hide programming details that provide object functionalities. An interface describes these functionalities in terms of methods and properties, providing a logical boundary between operation invocations and their implementations. Then an object is just a "server" of its own methods.

Objects in this respect closely resemble services with their plug and socket interfaces as implemented based on the above model-driven configuration service composition engine. Furthermore, similarities between the Object-Oriented paradigm and the Service Oriented paradigm as illustrated by this composition engine extend to a number of properties typical of objects and Object-Orientation. Referring to Figure 4, we draw a parallel. The concept of an ontology is fundamental to both paradigms. Any development is based on an ontology appropriate to the application domain in question. Based on this ontology in object-oriented terms use cases and scenarios are defined. These usually lead to a class diagram detailing the architecture to be implemented. This is analogous to the object hierarchy produced from the object-oriented model employed by model-driven configuration. Another common mechanism used to convey semantics related to the overall architecture and propagate best-practice design are design patterns. In order to achieve a certain type of composition in an efficient way (based on best practice) default knowledge is required. This knowledge is provided by composite service templates previously described. Design patterns directly correspond to such composite service templates.

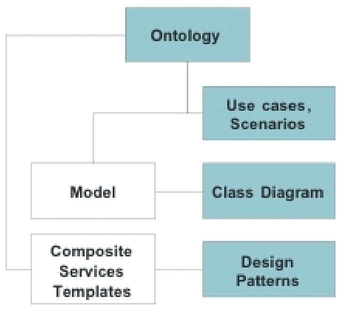


Figure 4: Relations between model driven configuration concepts and object orientation.

Based on these parallels further similarities can be established, see Figure 5. Elementary or composite service definitions directly correspond to classes. Categories of services, providing convenient ways of sorting large amounts of instances of services, are the equivalent of abstract classes, that describe common features but can not produce objects through instantiation. Constraints on the other hand are the equivalent of preconditions and post-conditions commonly used in object-oriented development.

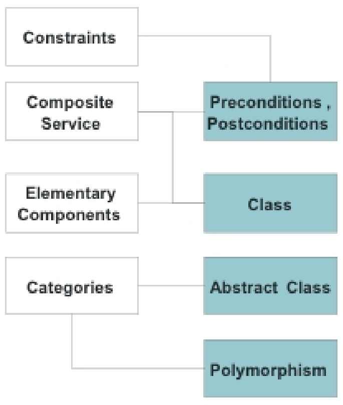


Figure 5: Additional relations between model driven configuration concepts and object orientation

More object related concepts can move into the service oriented world in order to enhance the technology and, perhaps, clarify the role and scope of web services. Here are the most immediate example of concept migration:

Inheritance. Two modes of inheritance are used in OOP: code inheritance and interface inheritance. Interface inheritance is the most immediate to apply to web services. Consider a payment service, which could be subtyped in a service with acknowledgment of receipt. In a workflow, the former could be substituted by the latter as it is guaranteed that the same port types are implemented in the subtyped service. Inheritance enables service substitution, service composition and it induces a notion of inheritance on entire compositions of services. Consider a workflow A built on a generic service and another one B with the same data and control links, but built on services which subtype the services of A. Could we say that B inherits from A or that B is a specialization of A?

Polymorphism. Both inclusion polymorphism and overloading can be extended to the service paradigm. A composition operation in a workflow may have different meanings depending on the type of the composed services. For example, composing a payment and a delivery service may have a semantics for which the two services run in parallel; on the other hand, the composition of two subtyped services in which the payment must be acknowledged by the payers bank and the delivery must include the payment transaction identifier have the semantics of a sequencing the execution of the services.

Composition. A formal and accepted notion of composition is currently missing in the SOC domain and, as just proposed, inheritance and polymorphism could induce such precise notions of composition over services. Some of the gaps left by standards which do not have a clear semantics, most notably, BPEL [3], could benefit from semantically funded definitions of composition.

4. INTERFACE INHERITANCE FOR SERVICE COMPOSITION

Composition is a central issue both in the object-oriented paradigm and in service oriented computing. By means of composition an entity can access other independent entities during the execution of its operations. On the other hand, the concept of inheritance, which is quite central in object oriented systems, does not have a relevant role in the service oriented paradigm.

In object oriented systems, the term *inheritance* is used to describe the mechanism allowing the derivation of a class C_2 from another one C_1. Class C_2, the inheriting class, is said to be a subclass of C_1. The subclass class will have to present the same external interface of C_1, in addition to its own public interface. In other words, it is possible to treat as object O_2 of class C_2 as if it is of class C_1: that is O_2 will accept the same messages of objects of class C_1.

The behaviour of C_2 could extend or limit the behaviour of C_1, but the important fact is that it is defined with respect to C_1 behaviour. One may distinguish between several forms of inheritance [5], but in our discussion we focus on inheritance for *specialization*; a class is defined in terms of specialization of an already existing one – this is expressed by the "is a" relationships. For instance, if we state that a TextWindow is a Window, we mean that the TextWindow has all the properties and behaviors of the Window, plus some additional property and/or behaviour.

Specialization usually implies a semantic coherence between the two classes. When this is true, C_2, that is, the specializing class or subclass, is also called a *subtype* of the class C_1. If semantic coherence is not granted the subclass will just have the same names as C_1 for public variables and methods, but the meanings attached to these interface elements can arbitrarily change. In other words, the subclass requires only a syntactical match, while the subtype guarantees also a semantical match between the involved classes.

The concepts of subclass/subtype are also related to a distinction commonly made between what is sometime referred to as “true” or “code” inheritance versus interface inheritance. The former is used when, besides presenting the same external interface, a class includes also the same code of the inherited class. As a consequence, a subclass formed via code inheritance will also be a subtype unless it explicitly overrides the behaviour of the inherited class. The latter term, interface inheritance, is used when a class has the same external interface of the inherited one, but it has no direct access to its code. In this case, a subclass becomes a subtype only when the behaviour of the inherited class is reproduced with the same semantics.

In terms of implementation, a simplifying model is to view inheritance as a special form of composition. Composition generally implies wrapping the interface of the included classes, and filtering the communication between these classes and the external world – the composition operation could be completely hidden. In the case of inheritance, the interface of the inherited class is added to the one of the inheriting class, letting the external world know of the relationship between the two classes. In addition, in the case of code inheritance, the operation of the inherited class will also be available.

Inheritance can be described as if the inheriting class incorporates (composes with) the inherited one, but without filtering the communication; the inherited class interface is directly accessible. An object of the inheriting class responds to the same invocations as an object of the inherited class. If the subclass is also a subtype, the results will also be the same. To think at inheritance (subtyping) as a form of composition which maintains the interface (behaviour) of the composed object, makes it easier to reason about similar concepts in the service world.

Before presenting the application of interface inheritance for service composition (see Section 4.2), in the next section we discuss the role of composition in the service oriented paradigm and its relationship with the similar concept in OOP.

4.1 Object composition versus service composition

A large amount of effort in research literature and in industry standards is devoted to service composition. As representatives of the approaches mentioned at the end of Section 3, we refer on one hand to authors focusing on designing the composition of service (e.g., [6, 33]) and on the other hand to authors defining how semantically annotated services can be automatically composed (e.g., [20]).

Service composition based on model driven configuration addresses the problem of creation of composite services during run-time. This is achieved through the iterative composition of elementary components into a composite service based on well-defined constraints. Connections between elementary services are realized based on the aforementioned plugs and sockets concept where composition dependencies (connections) that make use of an interface component are referred to as sockets and components that are matched by a socket are referred to as plugs. A composition created based on this process consists of a group of elementary services connected via their interfaces in order to produce a more complex effect defined to be the composite service. Consequently a composite service can synthesize its functionality out of the functionality of a number of other services, e.g., a location based weather forecast service that is composed out of a service providing positioning data and a service providing weather forecast information. This behaviour can be cleanly mapped to the type of composition employed in the context of object oriented development, where the composite service functions as an inheritor and composing elementary services play the role of the parents.

In comparison, composition in Object Oriented development is a design-time activity mainly dealing with statically designing the architecture of the system. To state that an object is composed of another one, means that in the class diagram a containment relationship between the two corresponding classes exists. In this relationship the containing object is able to use the contained one, possibly shielding it from other parts of the system (see Section 4.2).

An additional level of detail, related to composition in the object oriented world, is grounded in the difference between the abstract view of classes and the instantiation process, that is, the creation of the actual objects. A composition relationship between classes C_1 and C_2 will lead to the fact that an object O_1 (instance of class C_1) will contain an object O_2 (instance of class C_2). This result can be achieved in two radically different ways: exclusive or non-exclusive composition. In the former case, the instantiation of O_1 will create O_2, a new instance of C_2; when O_1 will be destroyed, O_2 will also be deleted. In the latter, non-exclusive, case, O_1 will make use of O'_2, an already existing instance of C_2; in this case, deleting O_1 will not affect O'_2.

Recapitulating, the main difference between service composition and composition of objects is that composite services are not statically designed, but rather are composed at run-time, as services providing the required supporting infrastructure are composed using dynamic discovery mechanisms. Consequently, the service paradigm provides the capabilities for dynamic, runtime composition instead of a statically planned architecture.

The dynamic nature of service composition has several consequences. A significant one is that negotiation and contractual agreements cannot be accomplished off-line, they have to be dealt with at run-time. The role of catalogues and the discovery mechanism have no counterpart in the world of objects and components.

Services demand a transition from static binding between objects or components that are to be integrated to the dynamic binding of services. From the point of view of the design there is the need of a transition from designing an architecture to designing the *enabling medium*, that is, the infrastructure for runtime composition.

Furthermore, a composite service functioning as the inheritor retains all the interfaces of its individual elementary components playing the role of the parents. This behaviour can be cleanly mapped to object-oriented inheritance mechanisms.

4.2 Interface inheritance for service composition

Interface inheritance allows to treat in the same way two elements of a composition relationship: with interface inheritance, a member of a composition can be substituted its inheritor (descendant). Interface inheritance for services guarantees the presence in the inheriting service of a specific interface: the inherited one.

An example is a service A designed for informing client services about conformance to certain policies, for instance, acceptance of a certain kind of credit card or availability of express shipping. A business process could then be designed in terms of requests to A and decisions based on its responses. If a second service B is built inheriting A interface: in addition to its own operations, it will respond to the A-like requests regarding card acceptance or shipping. Moreover, interface inheritance guarantees that the format of the requests accepted by B is the same as the ones of A. We can then substitute A with B in the business process. In addition, the service B may have further interface elements which do not affect the process.

We identify four different composition scenarios, which differ on the basis of the kind of operations performed and on the relationship between service interfaces. Table 6 summarizes the four scenarios, illustrated in Figures 7–9 and discussed in the reminder of this section. In Table 6 we use two categories for describing composition scenarios. Along the vertical dimension, we discriminate services according to the fact that the composing service presents (or not) to external applications the same interface elements as the composed services. On the horizontal dimension, we differentiate services according to the additional operations that are performed in addition to using the composed service. We define as *value-added* the operations that significantly change the nature of the operation of a composed service, while we define *pass-through* the operations that are only rearranging or reformatting data in addition to activating the composed service operation.

	Value-Added Operations	*Pass-Through Operations*
Same Interface	Sub-class composition	Sub-type composition
Different Interface	Opaque composition	Transparent composition

Figure 6: Composition and Inheritance.

In Figures 7–9, we represent a service with an oval in the diagrams and with capital letters in the text. Elements of the interface (that is, service operations) are represented by small shapes positioned on the oval boundary. Different shapes represent different operations, the same shape in two services indicates that the two services offer the same operation. In the text, interface elements are identified with i_x, i_y, and so on. Arrows represent requests or invocation of service operations. The + inside an oval of a service indicates that the service adds its own processing to a request, instead of just passing it to a composed service operation, possibly with some trivial data transformation. This second case is represented by a line connecting the interface element with the activation of the composed service. We also include simplified UML class diagrams, indicating the object oriented relationship from which we originate our description.

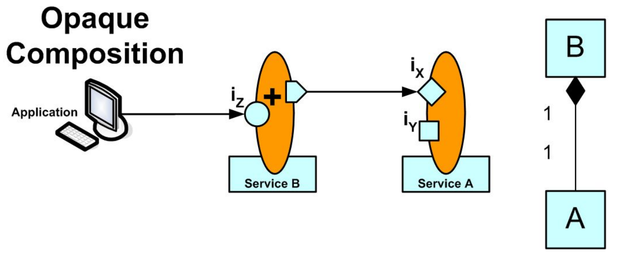


Figure 7: Opaque composition.

In the case of Opaque composition, see Figure 7, service B is composed by another independent services: A. The interface of service B is not related to the one of A. For an external application, there is no indication that B contains service A.

A request i_z to service B will be performed by activating operation i_x in service A. Besides requesting operations to A, B will perform additional work when it receives request i_z.

On the outside of B there is no notion of A operations.

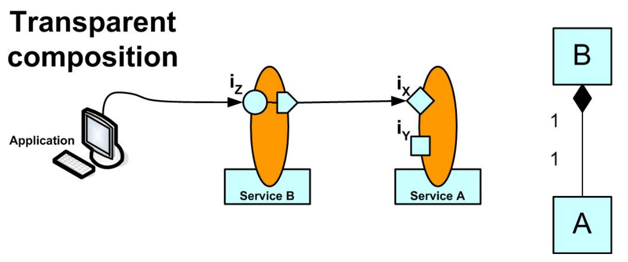


Figure 8: Transparent composition.

Transparent composition, see Figure 8, differs from Opaque composition because B does not process request i_z, but differently from the following cases, B's interface, i_z, is not the same as A's interface, i_x. For this reason, rearranging i_z data to match i_x format does not change the nature of this composition.

For instance, B could be a commercial service which is using A, a credit card validation service. Beside using a validation operation i_x of A, B could offer to external applications a validation operation i_z, using a different name and different parameters from i_x. Upon receiving request i_z, B will reorganize the request parameters and it will in turn ask A to perform i_x.

From the point of view of the external application, there is no connection between operation i_z of B and operation i_x

of A. They just happen to have a similar scope.

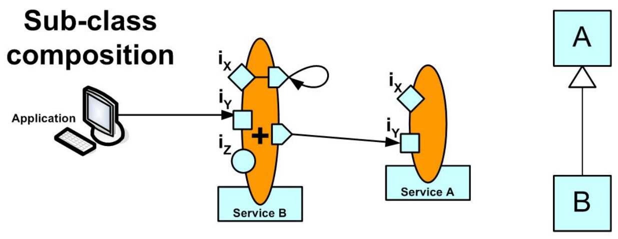


Figure 9: Sub-class composition.

With Subclass composition, see Figure 9, the role of inheritance starts to appear. Since service B inherits service A interface, it has to present to external applications the same interface as A, in addition to its own operations.

In Figure 9, B has i_x and i_y operations, with the same names and parameters as A operations. Since this is a subclass, there is no requirement to guarantee that B will produce the same results as the requests of the same these operations to service A. In fact, B could assign a completely different meaning to these operations.

Since B is not using A, there is no composition between the services. Nevertheless, from the point of view of an external application, B could be treated as an A, since having the same interface it will accept the same requests.

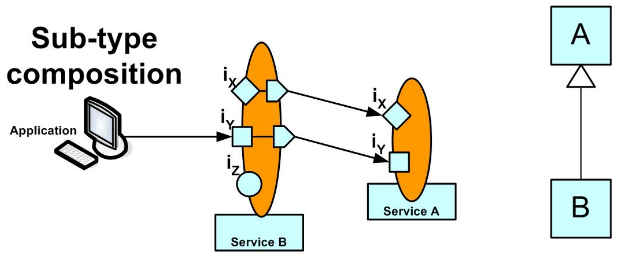


Figure 10: Sub-type composition.

As for Sub-class composition, in the case of Sub-type composition, (Figure 10), service B has the same interface as A, possibly with additional elements. The important difference is that B has to preserve the meaning assigned by A to its own operations.

One possible description of the case in Figure 10 is that B just receives the requests i_x and i_y, passing them on to A. It this way B guarantees that an accessing application will be able to treat B as if it was an A service, obtaining the same results.

Substitution of A with service B is possible also in the previous case, but without being semantically coherent.

4.3 Discussion

The widespread use of composition in systems based on service oriented architectures will ultimately lead to complex business models, relying on advanced service composition functionality. We suggest that the concept of composite services can be extended in a useful manner by allowing access to individual elementary services through interfaces exposed on the composite service. Examples scenarios where this type of extended composition would be useful are location based services (LBS). LBS typically require the integration of at least one service providing positioning data. Consequently every invocation of any composite LBS, like for instance a location based weather service, would also require the invocation of a service providing access to a positioning system, i.e., cellular positioning. If a user makes continuous use of composite LBS, a business model providing cost saving is to allow an already invoked composite LBS to participate in a new composition. In the new composition, the composite LBS would play the role of an elementary service, using only a subset of its functionalities. According to our model, the composite LBS would be used via the positioning system service interface only. In a different scenario, the motivation for this type of extended composition could be the provision of value-added services based on simpler versions provided by elementary components of a composition.

Such business models pose additional requirements for controlling the way the functionalities of the elementary services are composed and made accessible to the composite service consumer. Based on the concept of object-oriented inheritance, and of interface inheritance in particular, we propose a number of extended types of composition, supporting differentiated access modes to the functionalities and interfaces of elementary services.

Transparent, opaque, sub-type and sub-class composition can be compared to public, private and private protected interfaces in object-oriented terms. Much like object-oriented development makes use of such mechanisms to selectively expose interfaces to outside users or direct inheritors of a class, we propose access control mechanisms to achieve similar results when dealing with interfaces of elementary services participating in a more complex composition.

5. CONCLUDING REMARKS

The object oriented paradigm has a solid formal background and is a well-established reality of today's computer science. Service oriented computing is, on the other hand, a new emerging field, which tries to realize global interoperability between independent services. To meet this goal, service oriented technology will need to solve a number of challenging issues, such as how to manage service composition and orchestration. We have proposed a methodology based on model variant configuration by 'borrowing' concepts from the object oriented world. In particular, we have shown how the concepts of interface inheritance induce four forms of service composition.

Future investigation will be pursued in two directions. On the one hand, the utility of the approach will be tested by implementing a tool for designing compositions of services based on the proposed methodology. On the other hand, the added value of semantical enrichments of the interfaces will be investigated.

6. REFERENCES

[1] A. Barr and E. Feigenbaum, editors. *The Handbook of Artificial Intelligence*. Kaufman, 1981–82. Vols. 1–2.

[2] R. Barták. *Week of Doctoral Students (WDS99)*. MatFyzPress, 1999.

[3] BEA, IBM, Microsoft, S. AG, and Siebel. Business Process Execution Language for Web Services, 2003. `http://www-106.ibm.com/developerworks/library/ws-bpel/`.

[4] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. Automatic composition of e-services that export their behavior. In E. Orlowska, M. Papzoglou, S. Weerawarana, and J. Yang, editors, *Int. Conf. on Service Oriented Computing (ICSOC 03)*, LNCS, 2910, pages 43–58. Springer, 2003.

[5] T. Budd. *An Introduction to Object-Oriented Programming*. Addison Wesley, 2002. (3rd edition).

[6] F. Casati, S. Ilnicki, L. Jin, V. Krishnamoorthy, and M.-C. Shan. eFlow: a platform for developing and managing composite e-services. Technical report, Hewlett Packard, 2000.

[7] F. Casati and M. C. Shan. Definition, execution, analysis and optimization of composite E-Services. *Bullettin of the IEEE Computer Society Techincal Committee on Data Engineering*, 24(1):29–34, 2001.

[8] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems: Concepts and Design*. Addison Wesley, 2001. (3rd edition).

[9] R. Cunis, A. Günter, and H. Strecker. *Das PLACON-Buch. Informatik Fachberichte*. Springer, 1991.

[10] F. Curbera, W. Nagy, and S. Weerawarana. Web services: Why and how. In *Workshop on Obejcet Orientation and Web Services OOWS2001*, 2001.

[11] V. D'Andrea and M. Aiello. Services and objects: Open issues. In G. Piccinelli and S. Weerawarana, editors, *European workshop on OO and Web Service*, pages 23–29, 2003. IBM Research Report. IBM. Computer Science, (RA 220).

[12] S. Davis. *Future Perfect*. Addison-Wesley, 1987.

[13] P. Drucker. *The Practice of Management*. New York: Harper, 1954.

[14] I. Fikouras and E. Freiter. Service discovery and orchestration for distributed service repositories. In E. Orlowska, M. Papzoglou, S. Weerawarana, and J. Yang, editors, *Int. Conf. on Service Oriented Computing (ICSOC 03)*, LNCS, 2910, pages 59–74. Springer, 2003.

[15] I. Fikouras and F. Ramme. Service orchestration with generic service elements. In *Proceedings of the 6th International Symposium on Wireless Personal Multimedia Communications*, 2003.

[16] R. Glazer. Winning in smart markets. *Sloan Management Review*, 40:59–69, 1999.

[17] M. Heinrich. Ressourcenorientieres konfigurieren. *Künstliche Intelligenz*, 7(1):11–15, 1993.

[18] C. Huffman and B. E. Kahn. Variety for sale: Mass customization or mass confusion? Technical Report R98-111, Marketing Science Institute, 2004. `http://www.msi.org/msi/publication_summary.cfm?publication=491`.

[19] B. Kahn. *Dynamic relationships with customers: high-variety strategies*, volume 26. Sage, 1998.

[20] S. McIlraith and T. C. Son. Adapting Golog for composition of semantic web-services. In D. Fensel, F. Giunchiglia, D. McGuinness, and M. Williams, editors, *Int. Conf. on Knowledge Representation and Reasoning (KR2002)*, pages 482–493, 2002.

[21] B. Neumann. Configuration expert systems: a case study and tutorial. In H. Bunke, editor, *Artificial Intelligence in Manufacturing, Assembly and Robotics*. Oldenbourg, 1988.

[22] NOMAD. Ist-2001-33292. Project Web-Site: `http://www.ist-nomad.org`.

[23] B. Orriëns, J. Yang, and M. P. Papazoglou. Model driven service composition. In E. Orlowska, M. Papzoglou, S. Weerawarana, and J. Yang, editors, *Int. Conf. on Service Oriented Computing (ICSOC 03)*, LNCS, 2910, pages 75–99. Springer, 2003.

[24] M. P. Papazoglou and D. Georgakopoulos. Service-oriented computing. *Commun. ACM*, 46(10):24–28, 2003.

[25] M. Papazoglou et al. SOC: Service Oriented Computing manifesto, 2003. Working draft available at `http://www.eusoc.net`.

[26] F. Piller. *Kundenindividuelle Massenproduktion: Die Wettbewerbsstrategie der Zukunft*. Carl Hanser Verlag, 1998.

[27] F. Puppe. Expertensysteme. *Informatik Spektrum*, 9(1), 1986.

[28] P. Schnupp, H. Nguyen, and T. Chau. *Expertensystem-Praktikum*. Springer, 1987.

[29] W. Tank. Wissensbasiertes konfigurieren: Ein überblick. *Künstliche Intelligenz*, 7(1):7–10, 1993.

[30] E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.

[31] M. Tseng and J. Jiao. Mass customization. In G. Salvendy, editor, *Handbook of Industrial Engineering*, pages 684–709. Wiley, 2001.

[32] P. West, D. Ariely, S. Bellman, E. Bradlow, J. Huber, E. Johnson, B. Kahn, J. Little, and D. Schkade. Agents to the rescue? *Marketing Letters*, 10(3):285–300, 1999.

[33] J. Yang and M. Papazoglou. Web component: A substrate for web service reuse and composition. In *CAiSE*, pages 21–36, 2002.

Privacy-based Ranking of Web Services

Abdelmounaam Rezgui
Department of Computer Science, Virginia Tech
7054 Hayock Road, Falls Church, VA, USA
Phone: 1 703 538 8343
rezgui@vt.edu

Athman Bouguettaya
Department of Computer Science, Virginia Tech
7054 Hayock Road, Falls Church, VA, USA
Phone: 1 703 538 8403
athman@vt.edu

ABSTRACT
In this paper, we propose a novel algorithm for privacy-based Web service ranking. The proposed algorithm is *inference aware* and *history sensitive*. Inference awareness refers to the ability to make it futile for Web services providers to use inference mechanisms to derive non explicitly disclosed information about users. History sensitiveness refers to the ability to prevent services from violating users' privacy by exploiting their invocation history. The algorithm's overhead is minimized so that it can be integrated in any *online* service discovery mechanism.

Keywords
Privacy, semantic Web, Web services, inference control

1. INTRODUCTION
Preserving users' privacy was identified as a significant challenge almost as early as the dawn of the computer industry [7]. Since then, privacy has been an issue in almost every new wave of computer technologies or computing paradigms. With the Web revolution, privacy, or *Web privacy*, has come to the fore as a problem that poses a set of challenges fundamentally different from those of the pre-Web era.

The Web has been essentially an interactive environment. Most of its content is interpretable only by humans. Users' involvement is necessary for virtually any Web-based transaction. This has motivated a research direction that aimed at introducing more automation in the Web. A key milestone in that research was the introduction of the idea of the *semantic Web*. The intuition behind the vision of the semantic Web is to transform today's Web into an "intelligent" infrastructure where many of the time-consuming, user-initiated and user-monitored tasks become automated. In this vision, machines become much better able to process and understand the data that they merely display at present [2].

A technology that is widely expected to play a definite role in enabling the semantic Web is *Web services*. The W3C defines a Web service as "a software system designed to support interoperable machine-to-machine interaction over a network" [21]. A Web service exposes an interface through which it may be automatically invoked by Web clients. A Web service's interface describes a collection of operations that are network-accessible through standardized XML messaging [10]. Research on Web services aims at enabling the *Web service computing* paradigm. In this new paradigm, Web services publish their functionalities through service registries. Web clients access these registries to discover the description of Web services providing specific functionalities. They then invoke the services according to that description. Several standards have been developed to enable the use of Web services. These include WSDL [18] (for service description), UDDI [20] (for service discovery), and SOAP [17] (for service invocation).

1.1 Motivation
In today's Web, users generally decide on initiating a Web transaction based on their prior knowledge of the different parties involved in that transaction. In the envisioned semantic Web, software agents will replace users in initiating Web transactions and controlling their execution. In typical semantic Web transactions, Web services and agents will interact to carry out sophisticated tasks on behalf of users. In the course of this interaction, they may automatically exchange sensitive private information about these users. A natural result of this increasing trend towards less human involvement and more automation is that users will have less control over how their personal information is manipulated by software agents and Web services. This requires fundamental changes in how Web services are discovered and selected. Research on Web service discovery has largely focused on the aspect of discovering Web services that best suit some given *functionality* requested by a user. Far less attention was devoted to the issue of retrieving Web services based on quality of service (QoS) criteria, e.g., performance, cost, security, reputation, and privacy. A central problem that must be solved before achieving the vision of the semantic Web is to enable agents to automatically establish a privacy-based ranking of several Web services that may be invoked to accomplish a given task. Agents may then use such a ranking to autonomously determine, according to the current context, the most privacy preserving invocation scheme to accomplish the given functionality.

1.2 Contribution and Paper Organization
In this paper, we develop a novel algorithm for privacy-based Web service ranking. The proposed algorithm is (i)

user dependent, (ii) *inference aware*, (iii) *history sensitive*, and (iv) *computationally light*. User dependency is a natural requirement for any privacy-based selection of Web services. Inference awareness and history sensitiveness make it futile for Web services providers to use inference mechanisms on users' invocation histories to derive non explicitly disclosed information about those users. Finally, the proposed algorithm is computationally light so that it can be integrated in any *online* service discovery mechanism.

This paper is organized as follows. In the next section, we give a brief literature review. In Section 3, we introduce the key concepts of our solution. In Section, 4, we give the details of the proposed Web service ranking algorithm. Section 5 concludes the paper.

2. RELATED WORK

Although the need to preserve privacy in the envisioned semantic Web is widely recongnized, little research was actually devoted to the development of solutions achieving that objective. Before the concept of Web services was introduced, most of the research focused on the aspect of hiding the *real* identity of Web users when they access static Web content (e.g., Web pages) and Web-based applications (e.g., search engines) [13, 14]. Techniques that addressed this aspect included: Crowds [12], anonymizing tools (e.g., Anonymizer [1], cryptographic techniques [3], onion routing [6]), and aliases (personae) generators for Web users [5].

On the standardization front, a notable privacy standard is W3C's Platform for Privacy Preferences Project (P3P). The motivation behind P3P is to develop an industry standard that "enables Web sites to express their privacy practices in a standard format that can be retrieved automatically and interpreted easily by user agents" [22]. A Web site implementing P3P expresses its privacy policy in a machine-readable format. Its users may configure their browsers to automatically determine if the Web site's privacy policy reflects their personal needs for privacy. P3P, however, provides no technical mechanisms that guarantee that Web sites actually implement their stated privacy policy. Moreover, P3P is proposed as a standard to specify the privacy of Web sites and *not* Web services; it only automates the process of checking that users' privacy will *probably* not be violated when they access applications through a P3P-enabled Web browser. In standards for Web services, the issue of preserving privacy was often absent. For instance, the two standards WSDL and UDDI provide little or no support for privacy enforcement.

Some of the research aiming at solving the privacy problem in the semantic Web focused on the concept of *ontologies*. The importance of ontologies in building the semantic Web is widely recognized. In particular, they are a central builiding block in making Web services computer-interpretable [4]. This, in turn, enables the automation of the tasks of discovering, invoking, composing, validating, and monitoring the execution of Web services [11, 15]. Ontologies will also play a central role in solving the privacy problem in the semantic Web. Building a *privacy ontology* for the semantic Web is one of the several recent propositions to enable Web agents to carry out users' tasks while preserving their privacy [9]. A recent solution that used ontologies was proposed in [19]. The authors present a privacy framework for Web services that allows users' agents to automatically negotiate with Web services on the amount of personal information to be disclosed. In this framework, users specify their privacy preferences in different permission levels on the basis of a domain specific ontology based on DAML-S, an ontology for Web services specified using DAML-OIL [16, 15].

Our work is also related to *inference control* (IC). Techniques developed in the area of IC have been widely investigated in the context of statistical databases [23]. The objective of inference control in this context is to prevent intruders from compromising privacy using inference techniques that draw on data mining, record linkage, and knowledge discovery [8]. Our work aims at initiating research that: (i) leverages established database inference control techniques to the semantic Web context, and (ii) investigates new solutions that are specifically tailored to the semantic Web.

3. PRIVACY-BASED WEB SERVICE RANKING

In this section, we introduce a few concepts necessary to the description of our ranking solution.

Definitions

Definition 1 : The *privacy profile* for a user u is a set $\mathcal{P}_u = \{(a_1, f_1), (a_2, f_2), .., (a_n, f_n)\}$ where: a_i is an attribute (e.g., `Name`, `PhoneNumber`) and f_i is the *privacy sensitiveness factor* of that attribute, i.e., the importance that the user u associates to the attribute a_i. Without loss of generality, we will assume that all attributes are discrete and ordered, i.e., for any two values of v_1 and v_2 of attribute a_i, there exists a finite number of other values v_{k_1}, v_{k_l} that attribute a_i can take such that: $v_1 \leq v_{k_1} \leq .. \leq v_{k_l} \leq v_2$. We note the size of the interval of values that attribute a_i can take by: $\parallel a_i \parallel$.

Definition 2 : The *invocation history* $\mathcal{H}_u$ of a user u is a list of *invocation records* $\{ r_1, .., r_m \}$. Each record r_i in $\mathcal{H}_u$ records the event of invoking operation op_{ij} of service s_i by user u at time t. Each invocation record also captures the values of the input parameters provided by user u to service s_i in the invocation. An invocation record is represented by: $\{ (s_i, op_{ij}, (a^1_{ij}, v_1), .. (a^k_{ij}, v_k), t)* \}$ where v_l is the value of attribute a^l_{ij}. The size of the history $\mathcal{H}_u$ (number of its entries) is noted $\parallel H_u \parallel$.

Definition 3 : Let A_i be a subset of attributes and a_j be an attribute. a_j is said to be *inferable* from A_i if it is computationally possible to derive one *single* value for a_i from *any* instantiation of A_i. We note this by: $A_i \longrightarrow a_j$. For example, if $A_i \equiv \{$ `ZipCode` $\}$ and $a_j \equiv$ `City`, then: $A_i \longrightarrow a_j$.

Definition 4 : Let A_i be a subset of attributes and a_j be an attribute. a_j is said to be *potentially inferable* from A_i if it is computationally possible to derive a *finite* set of values for a_j from *any* instantiation of A_i. We note this by: $A_i \longrightarrow_p a_j$. For example, if $A_i \equiv \{$ `HighestDegree, JobTitle` $\}$ and $a_j \equiv$ `SalaryRange`, then: $A_i \longrightarrow_p a_j$. We note the cardinality of the set of values that attribute a_j can take for a given instatiation V_i of A_i by: $\parallel A_i \longrightarrow_p a_j \parallel_{V_i}$.

Definition 5 : The *universal inference function* $\mathcal{I}$ is a function defined as follows:

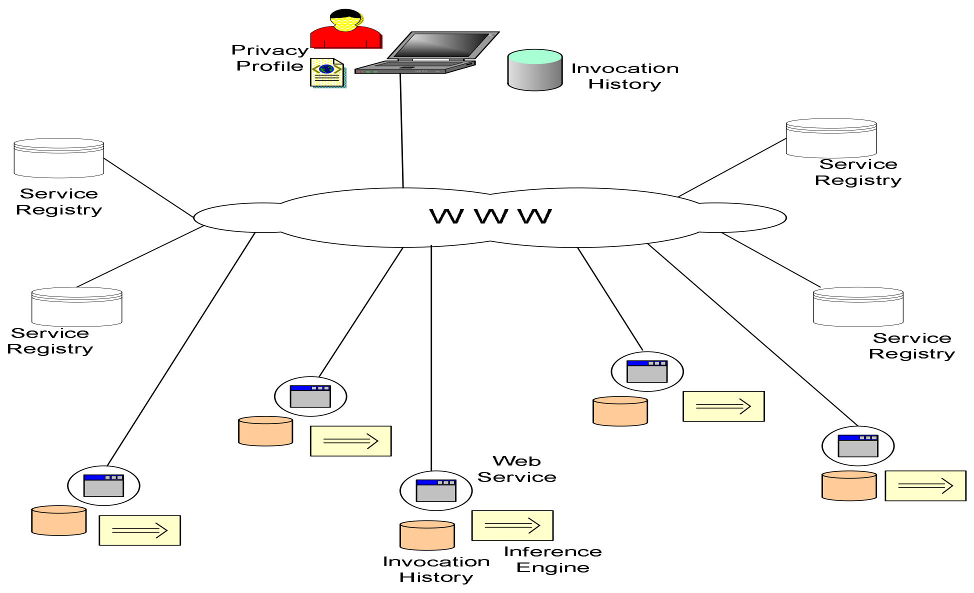


Figure 1: Service Ranking Principle

For each attribute subset A_i and attribute a_j:

$$\begin{cases} A_i \longrightarrow a_j & \Longrightarrow \quad \mathcal{I}(A_i, a_j) = 1 \\ A_i \longrightarrow_p a_j & \Longrightarrow \quad \mathcal{I}(A_i, a_j) = \alpha_{ij} \end{cases}$$

where α_{ij} $(0 \leq \alpha_{ij} \leq 1)$ is given by:

$$\alpha_{ij} = \frac{1}{max(\| A_i \longrightarrow_p a_j \|_{V_i})}$$

α_{ij} is called the *confidence* of the inference $\mathcal{I}(A_i, a_j)$. Consider the following examples:

1. $\mathcal{I}$({ `ZipCode` }, `City`) $= 1$
2. $\mathcal{I}$({ `FamilyName` }, `CountryOfOrigin`) $= 0.4$
3. $\mathcal{I}$({ `CountryOfOrigin` }, `NativeLanguage`) $= 0.8$
4. $\mathcal{I}$({ `HighestDegree, JobTitle` }, `SalaryRange`) $= 0.7$

In the first example, the confidence is 1 because the attribute `City` can deterministically be derived from the attribute `ZipCode`. The other examples illustrate cases where the value of an attribute may be derived, with *some* confidence, from the value(s) of one or more other attributes.

Closure of the Universal Inference Function

The closure of the universal inference function, noted $\mathcal{I}^*$, is derived from $\mathcal{I}$ by appliying the *transitivity inference rule.* This rule may be stated as follows:

- Initially, $\mathcal{I}^* := \mathcal{I}$
- If

$$\begin{cases} \mathcal{I}(A_1, a_1) = \alpha_1 \\ \mathcal{I}(A_2, a_2) = \alpha_2 \\ a_2 \in A_1 \end{cases}$$

then

$$\mathcal{I}^*(A_1 \cup A_2, a_1) := max(\mathcal{I}^*(A_1 \cup A_2, a_1), min(\alpha_1, \alpha_2))$$

Definition 6 : Let s_i be a Web service and op_{ij} an operation of s_i. The *input signature* of op_{ij}, noted $IS(op_{ij})$, is the set of its input parameters $\{a_1,\ a_2,\ ..\ ,\ a_k\}$.

Definition 7 : Let s_i be a Web service of which a user u has invoked one or more operations. The *invocation knowledge* that service s_i has of user u, noted $IK^u_{s_i}$, is the set of pairs $\{(a,\ v)\}$ where a is an attribute and v is the value of attribute a as collected by service s_i *directly* through an invocation from user u.

Definition 8 : The *potential knowledge* that service s_i has of user u, noted $PK^u_{s_i}$, is the information that the service s_i may *derive* on the user u by applying the closure inference function $\mathcal{I}^*$ on the attributes of $IK^u_{s_i}$. This knowledge may be represented as a set of pairs $\{(a,\ \alpha)\}$ where a is an attribute and α is the maximum confidence that service s_i has in the value of a after applying all possible rules of $\mathcal{I}^*$. For example, by applying the transitivity rule, we may derive that:

$$\mathcal{I}^*(\{\texttt{FamilyName}, \texttt{CountryOfOrigin}\}, \texttt{NativeLanguage}) = 0.4$$

We also may derive that (`NativeLanguage`, 0.4) $\in PK^u_{s_i}$.

4. PRIVACY-BASED RANKING

In this section, we first describe the general idea of our service ranking scheme and then present and evaluate the proposed ranking algorithm.

4.1 Ranking Principle

The basic idea of the proposed ranking scheme is based on the notion of *potential exposure* of users invoking Web services. Consider a service s_i with a *potential knowledge* $PK^u_{s_i}$ on user u. Let $\mathcal{P}_u \equiv \{(a_1,\ f_1),\ (a_2,\ f_2),\ ..,\ (a_n,\ f_n)\}$. The *potential exposure* of user u (from his/her own perspective) to the service s_i, noted by $\chi(u{,}s_i)$, is defined by:

```
Algorithm 1.
Input:
        - A set of Web services $S$
        - A user's privacy profile $\mathcal{P} \equiv ((a_1, f_1), (a_2, f_2), .., (a_n, f_n))$
        - The closure of the universal inference function $\mathcal{I}^*$
        - A History $\mathcal{H}$ that records the past service invocations of user $u$
        - An exposure threshold $\theta$
Output:
        - An ordered list of Web services $\Lambda$ where services are ranked according to their
          privacy preserving potential
begin

(0)     $\Lambda := \emptyset$;
(1)     for each service $s_i \in S$:
(2)                 $e :=$ Exposure $(\chi(u, s_i), s_i), \mathcal{P}_u, \mathcal{I}^*, op_{ij})$
(3)                 if ( $e > \theta$)
(4)                       then $S := S - \{s_i\}$;
(5)                 else
(6)                       $k$-Insert $(s_i, \delta)$ in the list $\Lambda$;
(8)                 endif
(9)     endfor
end
```

Figure 2: Web Service Ranking Algorithm

$$\chi(u, s_i) = \sum_{(a,\alpha)\in PK^u_{s_i}, (a,f)\in\mathcal{P}_u} \alpha.f$$

Example:

- $PK^u_{s_i} = \{$ (`Name`, 1), (`CountryOfOrigin`, 0.2), (`SalaryRange`, 0.7) $\}$

- $\mathcal{P}_u = \{$ (`Name`, 0.2), (`Address`, 0.9), (`CountryOfOrigin`, 0.8), (`SalaryRange`, 0.2) $\}$

In this case, the exposure of user u to service s_i is given by:

$$\chi(u, s_i) = 1 \ . \ 0.2 + 0.2 \ . \ 0.8 + 0.7 \ . \ 0.2$$

$\Box$.

The principle of the proposed algorithm (Figure 1) is as follows. Users will keep track of their own view of the potential knowledge that services already have about them. A user u may derive this value for service s_i from his/her invocation history $\mathcal{H}_u$. Assume that the user may invoke one of several services to achieve a given functionality. The ranking process precedes the service selection. It consists to compute the *anticipated* user's exposure for each alternative. Three parameters are required to compute the anticipated potential exposure that would result from invoking operation op_{ij} of service s_i: the user's privacy profile $\mathcal{P}_u$, the current potential exposure of user u to service s_i, $\chi(u, s_i)$, and the input signature of operation op_{ij}, $IS(op_{ij})$. The output of the ranking process is the list of candidate services ordered in the increasing order of their anticipated potential exposure.

4.2 Ranking Algorithm

In this section, we present our service ranking algorithm. The algorithm (Figure 2) takes as its input: (i) a set of Web services's, (ii) a user u's privacy profile $\mathcal{P}_u$, (iii) the closure inference function $\mathcal{I}^*$, (iv) the user's invocation history $\mathcal{H}_u$, and (v) a user-specified *exposure threshold* θ. The output of the algorithm is a list Λ of services ranked in the order of increasing anticipated potential exposure.

The function `Exposure` (Line 2) computes the anticipated potential exposure of the user u *if* service s_i is invoked. Let $\mathcal{P}_u$ be user u's privacy profile. For an operation op_{ij} with input signature $IS(op_{ij}) \equiv \{a_k\}_{1\leq k\leq p}$:

$$\texttt{Exposure}(\chi(u, s_i), \mathcal{P}_u, \mathcal{I}^*, op_{ij}) = \chi(u, s_i) + \sum_{\substack{a \in IS(op_{ij}) \\ (a, f) \in \mathcal{P}_u}} f \qquad (1)$$

The function k-`Insert` (Line 6) uses a sorted binary tree of k elements to keep track of the list of the current *best* k services. The list Λ has a maximum size k. The insertion of a service s_i whose anticipated potential exposure is lower than the service s_{max} with the maximum anticipated exposure already in the list Λ results in the substitution of s_{max} by s_i in the list Λ.

4.3 Analytical Evaluation

For the evaluation of our algorithm, we will use the following notations:
- s: number of Web services selected for ranking
- n: number of attributes in the user's privacy profile
- r: number of inference rules in $\mathcal{I}^*$
- o_{ij}: the number of attributes in the input signature of operation op_{ij}, i.e., $\mathcal{IS}(op_{ij})$=o_{ij}.

- l: the user-specified maximal size of the list Λ, i.e., $\parallel \Lambda \parallel = l$.

Line 1 of the algorithm is a loop of s iterations. In each iteration, the function `Exposure` adds the value of $\chi(u, s_i)$ to $\sum_{a \in IS(op_{ij}),(a,f) \in \mathcal{P}_u} f$ and checks each of $\mathcal{I}^*$ rules (see Eq. 1). This requires a computation of complexity $O(r.min(o_{ij}, n))$. Line 6 inserts an element in an ordered binary tree (the list Λ) of size l. This requires a processing cost of $O(l.log(l))$. Each of the other lines of the algorithm requires a constant processing cost. With the assumption that $o_{ij} << n$, the overall computational cost of the algorithm is: $O(s.r.n.l.log(l))$.

5. CONCLUSION

The vision of the semantic Web calls for tools that enable users to safely delegate much of the time consuming, user-initiated, and user-controlled tasks to software agents and Web services. A significant challenge is to automate the process of service selection. Automating this process would have a great consequences on users' privacy as agents and Web services will maniplate sensitive information on behalf of users. In this paper, we propose a privacy-based ranking algorithm that enables users and agents to *a priori* determine the extent to which their privacy may be at risk when several Web services may be candidate to accomplish the same functionality. The proposed algorithm may be used as a stand-alone solution for Web service ranking or incorporated in any type of infrastructure, e.g., for service search, composition, optimization.

6. REFERENCES

[1] Anonymizer. *http://www.Anonymizer.com*, 2002.

[2] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, May 2001.

[3] L. F. Cranor. Electronic Voting. *ACM Crossroads Student Magazine*, January 1996.

[4] D. Fensel and M. A. Musen. The Semantic Web: A Brain for Humankind. *IEEE Intelligent Systems*, 16(2), March-April 2001.

[5] E. Gabber, P. B. Gibbons, D. M. Kristol, Y. Matias, and A. Mayer. Consistent, Yet Anonymous, Web Access with LPWA. *Communication of the ACM*, 42(2), February 1999.

[6] D. Goldschlag, M. Reed, and P. Syverson. Onion Routing. *Communication of the ACM*, 42(2):39–41, February 1999.

[7] L. J. Hoffman. Computers and Privacy: A Survey. *ACM Computing Surveys*, 1(2):85–103, 1969.

[8] Josep Domingo-Ferrer, editor. *Inference Control in Statistical Databases: From Theory to Practice*. LNCS 2316. Springer, 2002.

[9] A. Kim, L. J. Hoffman, and C. D. Martin. Building Privacy into the Semantic Web: An Ontology Needed Now. In *Proc. of the Semantic Web Workshop*, May 2002.

[10] H. Kreger. Web Services Conceptual Architecture. *Report of IBM Software Group*, May 2001.

[11] S. McIlraith, T. C. Son, and H. Zeng. Semantic Web Services. *IEEE Intelligent Systems*, 16(2), March-April 2001.

[12] M. K. Reiter and A. D. Rubin. Anonymous Web Transactions with Crowds. *Communication of the ACM*, 42(2), February 1999.

[13] A. Rezgui, A. Bouguettaya, and M. Eltoweissy. Preserving Privacy in the Web: Facts, Challenges and Solutions. *IEEE Security & Privacy*, 1(6), November-December 2003.

[14] A. Rezgui, A. Bouguettaya, and M. Eltoweissy. SemWebDL: A Privacy Preserving Semantic Web Infrastructure for Digital Libraries. *Intl. Journal of Digital Libraries (to appear)*, 2004.

[15] The DAML Services Coalition. DAML-S: Semantic Markup for Web Services.

[16] The DAML Services Coalition. DAML-S: Web Service Description for the Semantic Web.

[17] The World Wide Web Consortium. *Simple Object Access Protocol (SOAP)*. http://www.w3.org/TR/soap.

[18] The World Wide Web Consortium. *Web Services Description Language (WSDL) 1.1*. http://www.w3.org/TR/wsdl.

[19] A. Tumer, A. Dogac, and H. Toroslu. A Semantic based Privacy Framework for Web Services. In *WWW'03 workshop on E-Services and the Semantic Web (ESSW'03)*, May 2003.

[20] UDDI. *Universal Description, Discovery and Integration*. http://www.uddi.org.

[21] W3C. Web Services Architecture. *W3C Working Draft*, August 2003.

[22] T. W. W. W. C. (W3C). *The Platform for Privacy Preferences 1.0 (P3P1.0) Specification*, April 2002.

[23] L. Wang, S. Jajodia, and D. Wijesekera. Securing OLAP Data Cubes Against Privacy Breaches. In *Proc. of the 2004 IEEE Symposium on Security and Privacy*, pages 161–, May 09-12 2004.

Contact Author
Enrico Mussi
Politecnico di Milano
Piazza Leonardo da Vinci, 32
20133 Milano, Italy
Email: mussi@elet.polimi.it

Authors
Luciano Baresi, Devis Bianchini, Valeria De Antonellis, Florian Daniel, Andrea Maurino, Stefano Modafferi, Barbara Pernici.

Topic of interest
Workflow technologies & services, Service modeling and representation languages.

Keywords
Service-oriented computing, Service discovery, service selection

Abstract

Service oriented computing is becoming the standard paradigm to support the creation of applications composed of services selected from a registry. Nowadays, we are assisting to the proliferation of standardized approaches to describe such services, but there is the general agreement of distinguishing between the general characteristics of services and the characteristics associated with service invocation. In many cases, the selection of services is static and based on matching techniques to retrieve the most appropriate service.

The paper presents the MAIS architecture to provide highly adaptive services in a mobile and interactive environment and we focused on service selection and invocation, context-aware orchestration and mechanisms for managing user interaction in a service-oriented architecture. We propose adaptivity at different levels: at process level, during the selection of a concrete service, and also at end user level. Selection is based on suitable ontologies and considers the actual context and user characteristics to retrieve the most suitable services. The paper describes the main components of the architecture and exemplifies them on a simple process for a shipping company.

Provisioning of Complex Adaptive Services

L. Baresi, F. Daniel, A. Maurino,
S. Modafferi, E. Mussi, B. Pernici
Politecnico di Milano
P.zza L. da Vinci 32
20133 Milano, Italy
mussi@elet.polimi.it

D. Bianchini, V. De Antonellis
Università degli Studi di Brescia
Via Branze 38
25123 Brescia, Italy

ABSTRACT

Service oriented computing is becoming the standard paradigm to support the creation of applications composed of services selected from a registry. Nowadays, we are assisting to the proliferation of standardized approaches to describe such services, but there is the general agreement of distinguishing between the general characteristics of services and the characteristics associated with service invocation. In many cases, the selection of services is static and based on matching techniques to retrieve the most appropriate service.

The paper presents the MAIS architecture to provide highly adaptive services in a mobile and interactive environment and we focused on service selection and invocation, context-aware orchestration and mechanisms for managing user interaction in a service-oriented architecture. We propose adaptivity at different levels: at process level, during the selection of a concrete service, and also at end user level. Selection is based on suitable ontologies and considers the actual context and user characteristics to retrieve the most suitable services. The paper describes the main components of the architecture and exemplifies them on a simple process for a shipping company.

Keywords

Service-oriented computing, Service discovery, service selection

1. INTRODUCTION

The emerging paradigm of service-oriented computing supports the creation of applications by composing services selected among a variety of available services with different characteristics. Services may be invoked directly by the application in which they are used. Essential to this paradigm is the description of services using a standardized approach; in the literature, several proposals of service description languages have been made, such as WSDL, of service ontologies, such as in AgFlow [25], of semantic web services [1], separating general characteristics of services from the characteristics related to service invocation. In the above mentioned approaches, service selection is generally static, assuming matching techniques to retrieve the most appropriate services. In VISPO [2], authors introduce the concept of concrete and abstract services in the context of process definition, allowing the designer to specify the process in terms of abstract services and then providing an invocation environment to select the most appropriate concrete service. During selection and execution, the availability of the selected process is evaluated and mechanisms for substituting concrete services whenever they are not available are provided.

However, in both VISPO and AgFlow, where service unavailability is considered at runtime, the assumption is that, beyond unavailability of services, the context of invocation is always the same. This assumption cannot be considered valid anymore in applications running in a highly variable environment in terms of both the architecture and its components. In such environments, for example in the case of mobile information systems [15], services invocation may vary depending on their availability over the network, on parameters of devices on which they are invoked and on the characteristics of the networking infrastructure. In addition, services might be used in the process several times and their execution environment might change over time.

The goal of this paper is to propose the MAIS architecture and its mechanisms for designing and executing complex services exploiting adaptivity. We propose adaptivity at different levels: at the process level, at the level of selection of a concrete service for a given abstract service and at the level of interface between users and the platform. We support service selection with an enhanced UDDI registry, storing descriptions of abstract and concrete services and including information about quality parameters on the provider side. The proposal has been developed in the MAIS (Multichannel Adaptive Information Systems) Project [23].

The rest of this paper is organized as follows: Section 2 introduces the scenario for our running example, a mobile information system for a shipping company in order to provide motivations for the aspects discussed in the rest of the paper. Section 3 describes the MAIS functional architecture, focusing on orchestration and concrete services selection and invocation. It also introduces the basic services ontology of the MAIS Registry. Section 4 exemplifies the behavior of the functional architecture with respect to the running example and discusses a mechanism for decoupling service invocation from the design of a user environment in terms of its interaction with the system, based on an extension of the WebML

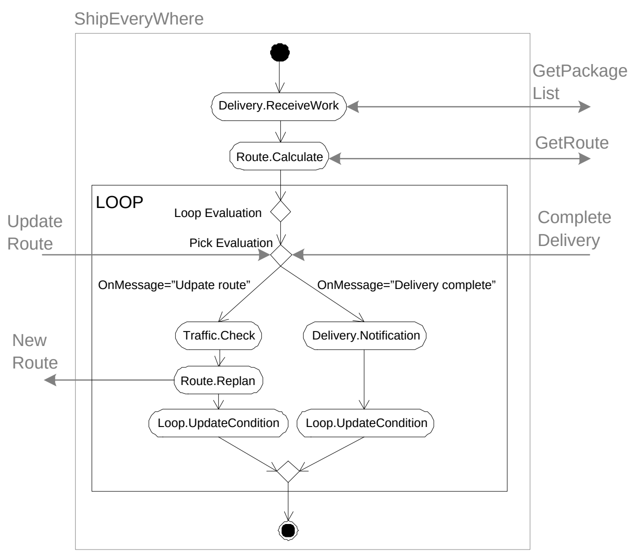


Figure 1: Workflow of shipping assignment and execution phases

model [10]. Section 5 discusses our proposal in relation to the state of the art. In Section 6 conclusions are reported and future work is anticipated.

2. THE SHIPPING COMPANY

This section presents the scenario of the running example used throughout the paper. It describes the typical problems of a shipping company that wants to optimize the delivery of packages. We concentrate on a simplified version of the process to deliver goods and imagine that the need for optimization and adaptation of the delivery procedure to the context leads to enacting the process in several different ways.

ShipEveryWhere, our shipping company, has a unique process to support the delivery of all packages. For simplicity, we consider a single item and we also assume that the process starts after assigning the item to the best vehicle. The process includes the creation and update of the route followed by the vehicle. Figure 1 shows the workflow model of the process that oversees all the phases and is inspired by basic BPEL4WS [11] primitives. Notice that we use a dot notation to name activities: the first part identifies the service, while the second part specifies the operation.

The process starts with the assignment of the task to the vehicle that carries the item (service *Delivery*, operation *ReceiveWork*). The *ShipEveryWhere* control center, through an appropriate user interface, assigns the item and related delivery information to the selected driver. According to the destination and dimension of the item, the vehicle can be a truck, for destinations longer than 200 kilometers, a van, for destinations between 10 and 200 kilometers, and bulky packages, and a motorbike for close destinations, i.e., less than 10 kilometers and small boxes. Each vehicle is equipped with a device to interact with the control center via a GPRS interface. In particular, each truck hosts a laptop; vans have PDAs and motorbike drivers use a smart-phone. Vehicles are equipped with different devices because of the different uses and the available room on board. This choice, however, implies that the enactment of the process varies and that it must cope with the device on the actual vehicle.

After the assignment phase, the driver calculates the best route to deliver the item (service *Route*, operation *Calculate*). This activity varies according to capabilities of the device hosted by the vehicle and can be done in different ways. For example, the control center can send required data, the vehicle can use local (context-dependent) services to discover traffic conditions and calculate the best option, or drivers can decide based on their own experience.

While driving towards destination, the driver can either notify the control center that the item is delivered (service *Delivery*, operation *Notification*) or request the current situation of traffic conditions and consequently replan the route. The *UpdateRoute* operation can be required by the driver, in case of heavy traffic on the selected route, or by the control center to notify the driver of congested traffic conditions on the route (message *UpdateRoute* of *Pick* activity). The selection of the actual services that detect traffic conditions and replan the route depends on the driver's profile (e.g. the used device) and the specific context in which the request is placed (e.g. the availability or absence of a GPRS network can affect the set of available alternatives). If the driver is not able to connect to the control center, because the bandwidth is too low, he can connect to the *TIER* service (**T**raffic **I**nformation on **E**uropean **R**oads). If the vehicle cannot use GPRS channel (or if the driver declares in his profile that he does not want to pay for it), the replanning must be done locally (that is, manually) with the information on board or by using the driver's experience. In this last case, data must be supplied by the driver by means of a special-purpose user interface.

3. FUNCTIONAL ARCHITECTURE

This section introduces the MAIS functional architecture, its components and the relationships among them.

Before doing this, we must set the jargon used in the paper and clarify that we distinguish between:

- *abstract services*, which are services that cannot be invoked directly and for which only abstract aspects, like functional interface and QoS levels, are described; implementation details are omitted; the functional description is expressed using the abstract part of the WSDL specification, in terms of the operations that the service performs, the input values it requires for the execution and the output values it produces after execution; constraints on input and output values can be specified; the quality of service is expressed by means of a set of standard quality dimensions, imposed by the channels used for service delivery and guaranteed by the service provider; each dimension is described by a name and a range of admissible values (see [18]);

- *concrete services*, which are services directly invocable, which inherit the abstract functional interface and the QoS levels of an abstract service; a concrete service can extend the functional interface and QoS description with implementation specific details (i.e., access protocols, QoS values); the concrete services are clustered on the basis of their functional similarity (see [4]), evaluated on the WSDL abstract interfaces; abstract services are associated to the clusters of concrete services and their interfaces are representative of the interfaces of the concrete services in the clusters; in particular, the set of operations in the interface of the abstract service are only those common to all the concrete services in the cluster; the designer can possibly force further capabilities that are considered to be relevant for the cluster, for example, because they are present in most concrete services of the cluster; in all cases, proper mapping rules between the capabilities in the abstract service and those in the corresponding concrete services must be defined; they are defined on operation names and on I/O entity names; furthermore, in our framework concrete services are distinguished into two subcategories:

  – *simple concrete services*, which are concrete services that do not use the orchestration and substitution functionalities of the MAIS architecture; all these operations are made by the provider and are hidden for our framework and for users;
  – *complex concrete services*, which are concrete services containing an abstract process definition which will be instantiated and executed using the orchestration and substitution functions offered by our framework.

  Contextual information is given in terms of conditions under which the service is provided and are used to further refine concrete services presented to the end users; in particular, it refers to the *Channel* used for service provisioning, the *Location* where it is used and the *Time* at which it is executed;

- *MAIS services*, which are generic services offered by the MAIS framework to end users; this definition masks the framework complexity, hiding service classification to end users, who work only with MAIS services while the management of the different kind of services is performed by the framework.

There are two kinds of actors which interact with the MAIS architecture:

- *designers*, who define processes contained into the complex concrete services; designers browse the registry to search for MAIS services, select them, and compose processes with the selected services;
- *end users*, who invoke the MAIS services; end users interact with the platform for searching and invoking services.

3.1 MAIS architecture

Figure 2 shows the MAIS functional architecture and the relationships among its modules. The architecture is composed by six modules that cooperate to provide and manage complex services in a context aware manner.

The *Platform Invocator* represents the point of contact between the *User Environment* and the MAIS architecture and hides the complexity of the architecture. Its interface exports a set of operations, which allow interacting programmatically with the architecture, performing operations like: i) searching published services in the *MAIS Registry* ii) executing the chosen services and iii) managing the interaction with the *User Environment* during the execution of a complex service. For this reason, end users interact through the *User Environment* and not directly with the *Platform Invocator*. The *User Environment* provides the graphical interface for end users who want to interact with the MAIS architecture. What distinguishes this module from a simple static GUI is the ability to dynamically generate the user interface with respect to the context in which end users are (i.e., devices, available communication protocols, user profile). This module is realized using *WebML* (see Appendix), which is a well established visual notation for the conceptual design of data-intensive Web applications and has recently been extended with new primitives also supporting the integration of Web services and thus suits our needs.

The information about the context is taken from the *MAIS Reflective Architecture Interface*. This module represents the access point to the reflective middleware and allows other modules to observe and modify the context of the execution and capture relevant events from the reflective middleware. These events provide useful information to the architecture for the provisioning of adaptive services (i.e., QoS degradation or battery level of a mobile device).

Once end users select and invoke a service using the *User Environment*, the management of such an execution is performed by the core modules of the MAIS architecture. These modules are: i) the *Process Orchestrator* for managing the execution of complex concrete services and ii) the *Concrete Service Invocator* for instantiating the services and executing the calls of concrete service operations.

The *Process Orchestrator* manages the state of the process and, step by step, interacts with the *Concrete Service Invocator* for invoking each operation specified in the definition of the workflow. The *Process Orchestrator* invokes abstract operations using abstract parameters; it is up to the *Concrete Service Invocator* to translate abstract parameters into concrete ones and invoke the concrete operation compatible with the abstract one.

The *Concrete Service Invocator*, which is in charge of managing the invocation of services, can:

- Start the invocation of concrete services. When the *Platform Invocator* asks for a service invocation, the *Concrete Service Invocator* invokes the correct concrete service after the interaction with the *MAIS Registry.*

- Invoke abstract operations. This is a sophisticated functionality used by the *Process Orchestrator* for invoking abstract operations. An abstract service cannot be invoked and we need to select concrete services. During this phase, called *link phase*, the *Concrete Service Invocator* accesses the *MAIS Registry* for finding concrete services and evaluate their affinity with respect to the abstract service (i.e., the request). Once a compatible concrete service is chosen, the *Concrete Service Invocator* proceeds by invoking the concrete operation. The *Concrete Service Invocator* receives as input the parameters of the abstract operation, translates them into the concrete ones, invokes the concrete operation and then translates the concrete output parameters into abstract ones. The translation of parameters is performed by using wrappers registered in the *MAIS Registry.*

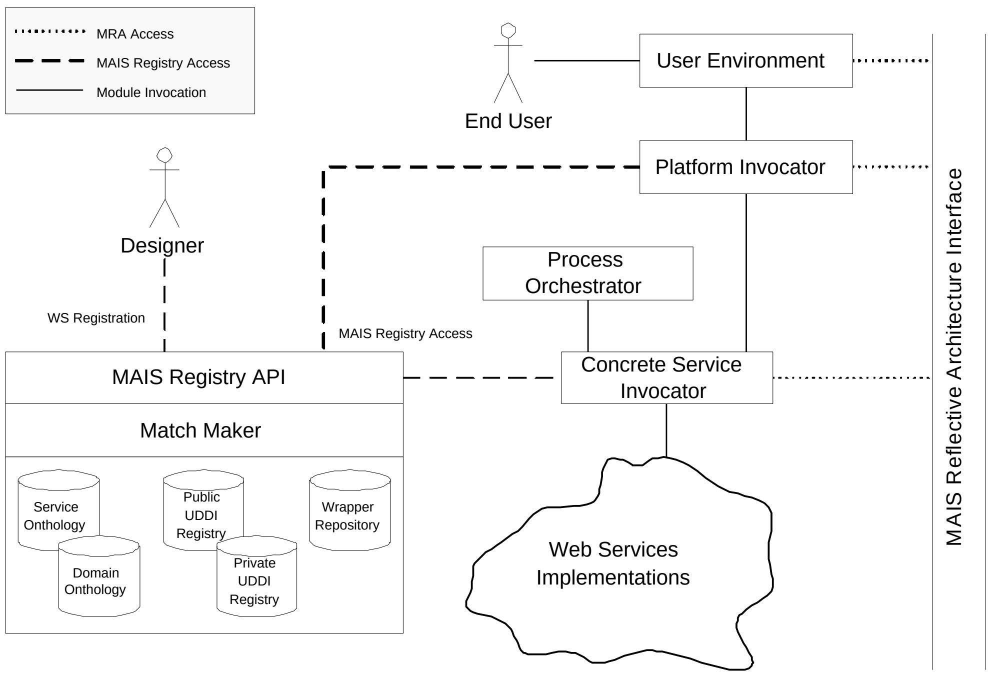


Figure 2: MAIS functional architecture

- Invoke concrete service operations. This invocation is performed by accessing directly the concrete implementations of services and invoking the concrete operations by passing concrete parameters.

The last module of our architecture is the *MAIS Registry*. This module contains suitable descriptions of all published services, along with other auxiliary information.

The MAIS registry is composed of: a *UDDI registry*, where services are registered with associated keywords, a *domain ontology*, where semantic information for service input/output annotation is maintained, and a *service ontology*, where services and semantic relationships among them are organized in two different layers (concrete layer and abstract layer), as explained later. The relevance and benefits of an architecture, which combines UDDI registries with ontologies, have been already motivated in [14] for semantic service matchmaking.

The service ontology is organized in two layers: in the *concrete layer*, concrete services are grouped into clusters according to identified semantic similarity relationships; in the *abstract layer*, abstract services are related to each other by means of semantic generalization and/or part-of relationships. An *association link* is maintained between each abstract service and the corresponding cluster.

During process execution, the *MAIS Registry* is directly accessed by the *Concrete Service Invocator* to find the concrete services that must be invoked. Given an abstract service, the *MAIS Registry* searches the available concrete services associated to the abstract one, applies mapping rules and proposes services to the *Concrete Service Invocator*. At this point, context and quality requirements are checked to filter the proposed concrete services.

The MAIS architecture also comprises a *Match Maker* component, which provides the functionalities for publishing and retrieval services on which the *MAIS Registry API* is based. During the publication phase, the *Match Maker* analyzes the published service description, compares it with service descriptions that already exist in the *MAIS Registry* using a set of techniques for affinity evaluation and updates the contained UDDI registry and the ontologies. Furthermore, the *Match Maker* acts as search engine for browsing the registry and the ontologies, in order to retrieve all the concrete services compatible with an abstract one.

4. RUNNING EXAMPLE

After introducing the main components of the MAIS architecture, this section exemplifies their behavior with respect to the shipping company example illustrated in Section 2.

The execution of the process specification depicted in Figure 1 is up to the *Process Orchestrator*. Its main tasks are: i) deciding when to invoke an abstract operation and ii) controlling the *link phase* of the *Concrete Service Invocator* to bind the choice of concrete services to the execution context.

In this example, the choice of which operation to invoke is very simple and only depends on the process specification. The orchestrator selects an abstract operation and uses the *Concrete Service Invocator* to invoke it. For instance, in the shipping company example, the orchestrator waits until a message triggers the activity *ReceiveWork*. When this happens, the orchestrator invokes the abstract operation *Calculate*, notifies the calculated plan to the driver and then waits for other incoming messages. If an *UpdateRoute* message is notified, the orchestrator invokes the abstract operations *Check* and *Replan* and then waits again. If a *CompleteDelivery* message is notified, the orchestrator invokes the abstract operation *Notification* and terminates the process.

More interesting is the managing of the *link phase*. There are various concrete services that provide operations for checking, calculating or replanning and the selection of the suitable concrete services depends on the execution context, like, for example, the position of the vehicle. A driver continuously changes his position and every time that the orchestrator needs to invoke an abstract operation for checking traffic or planning the route, it must be sure that the concrete service that performs such operation covers the geographic area where the vehicle is. This is done by forcing the *link phase* before invoking the operations *Check* or *Replan*.

Initially, the *Process Orchestrator* requires the abstract service *Route* to be linked, which contains the abstract operation *Calculate*. The *Concrete Service Invocator* searches the *MAIS Registry* for selecting concrete services that are compatible to the abstract service *Route*. This search is performed by considering constraints derived from the execution context, like the geographic position of the vehicle or the minimum GPRS bandwidth required. For example, if we only consider geographical constraints, the *Concrete Service Invocator* would select concrete services that offer a route service that covers the geographic area in which the driver is. After the search phase, the *Concrete Service Invocator* chooses the most suitable service among selected ones. This can be done, for example, by choosing the service that offers the widest GPRS bandwidth.

After executing the *link phase*, the *Process Orchestrator* can invoke the operation *Calculate* on the linked abstract service by sending the invocation request and related abstract parameters to the *Concrete Service Invocator*. The *Concrete Service Invocator* transforms the abstract parameters into concrete parameters by means of proper wrappers and then invokes the operation on the previously selected concrete service. Returned parameters are also converted by means of the same wrapper and sent back to the *Process Orchestrator*.

If the *Process Orchestrator* invokes the operation *Replan* on the previously linked abstract service *Route*, the *Concrete Service Invocator* performs such an invocation on the concrete service already selected. If such service is unavailable, a concrete service belonging to the same set of selected concrete services must be chosen. This behavior implies that, if the *Process Orchestrator* needs to use services with a particular geographical position, it has to perform the *link* operation every time that the vehicle changes its position.

A particular case of the *link process* concerns the abstract service *Delivery*, which is used by the orchestrator for invoking the operation *Notification*. If we suppose that there is only one concrete service that realizes such an operation (i.e. the *ShipEveryWhere* concrete service) there is no need for the *Concrete Service Invocator* to search the *MAIS Registry* for selecting the proper concrete service. The search is avoided by the *Process Orchestrator* that performs a special *link* over the *Concrete Service Invocator* to permanently bind the abstract service *Delivery* to the concrete service *ShipEveryWhere*.

As stated before, besides the functionality related to service invocation, the *Concrete Service Invocator* is responsible for delivering messages between the *Process Orchestrator* and the *Platform Invocator*. When the process begins, the driver must be informed about the task and subsequently about the route he has to follow. This is done by the *Process Orchestrator* that uses the functionality of the *Concrete Service Invocator* for delivering messages to the *Platform Invocator* and implicitly to the driver. The same thing is performed by the driver who uses the *Platform Invocator*, via the *User Environment*, to notify *UpdateRoute* or *CompleteDelivery* messages to the orchestrator.

The *Platform Invocator* represents the access point to the MAIS architecture. It notifies allocated tasks and related routes to the driver; this is done using an *activity list*. The *Platform Invocator* manages a list which contains all the activities (tasks and advices, for example) assigned to drivers.

When a driver accesses the architecture via the *User Environment*, he reads the assigned task, views the assigned route and performs the delivery to the correct destination. If during the delivery process the driver decides to recalculate the route, he has to notify the decision to the control center by sending an *UpdateRoute* message via the *User Environment*. The *Process Orchestrator* receives this message and reacts consequently. The same thing must be done by the driver when he completes the delivery.

Moving to the MAIS Registry, Figure 3 shows a portion of the service ontology of *ShipEveryWhere*. We have four abstract services associated with the corresponding clusters of concrete services. We suppose that:

- the *Concrete Service Invocator* receives a request of a service to replan route or to obtain traffic information from truck-A with a laptop that uses the GPRS network and requires a high bandwidth (greater than 100Kbps);
- the location scenario is the European one (context information).

The *Concrete Service Invocator* exploits the functional matching mechanism to find the abstract services *Route Planning* and *Traffic*. In the first case, it has to choose between the concrete services *PlanRoute* and *Easy Europe Travel*: for both these services the location is acceptable and both of them are provided on the GPRS network, but only the second one has an acceptable bandwidth value. So only the concrete service *Easy Europe Travel* is returned to the *Concrete Service Invocator*. The selection of a concrete service for the abstract service *Traffic* is similar. Suppose now that the same request is sent from the motorbike-D, which is equipped with a smartphone that uses the UMTS network. The connection to the concrete services *Easy Europe Travel* and *SocietàAutostrade* is not possible, since they are only provided on GPRS networks. On the other hand, services *PlanRoute* and *TIER* are also supplied on UMTS networks and are proposed to the *Concrete Service Invocator*. Finally, let us suppose that we need a planning with cost evaluation: in this case, functional requirements concern a planning operation that returns the cost as output parameter. In our example, the *Concrete Service Invocator* uses the functional matching algorithm to obtain the abstract service *Planning with Cost*, for which, however, only the concrete service *Euro Itinerary* is acceptable, since for the service *Milan-Rome Map&guide* the location is too restrictive. So, if a GPRS network is not available, we have two solutions: the *Concrete Service Invocator* does not return concrete services as result or it exploits the *is-a* relationship and presents as result the service *PlanRoute* associated with the more general abstract service *Route Planning*.

This example shows how the service ontology can be exploited to enhance adaptive service provisioning, starting from searching abstract services with required functional capabilities, then locating groups of suitable concrete services

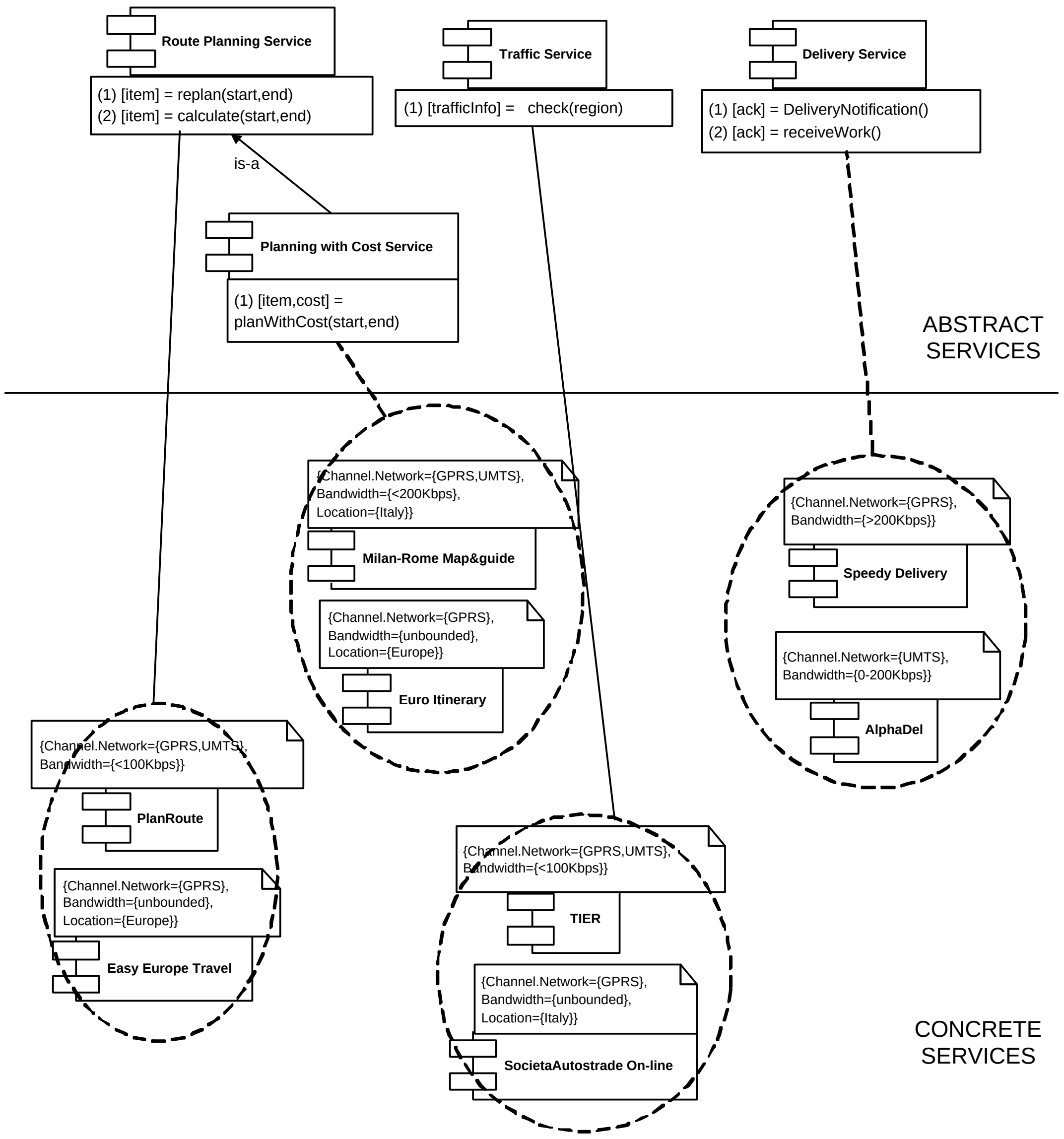

Route Planning Service
(1) [item] = replan(start,end)
(2) [item] = calculate(start,end)
Traffic Service
(1) [trafficInfo] = check(region)
Delivery Service
(1) [ack] = DeliveryNotification()
(2) [ack] = receiveWork()
is-a
Planning with Cost Service
(1) [item,cost] = planWithCost(start,end)
ABSTRACT SERVICES
{Channel.Network={GPRS,UMTS}, Bandwidth={<200Kbps}, Location={Italy}}
Milan-Rome Map&guide
{Channel.Network={GPRS}, Bandwidth={unbounded}, Location={Europe}}
Euro Itinerary
{Channel.Network={GPRS}, Bandwidth={>200Kbps}}
Speedy Delivery
{Channel.Network={UMTS}, Bandwidth={0-200Kbps}}
AlphaDel
{Channel.Network={GPRS,UMTS}, Bandwidth={<100Kbps}}
PlanRoute
{Channel.Network={GPRS}, Bandwidth={unbounded}, Location={Europe}}
Easy Europe Travel
{Channel.Network={GPRS,UMTS}, Bandwidth={<100Kbps}}
TIER
{Channel.Network={GPRS}, Bandwidth={unbounded}, Location={Italy}}
SocietaAutostrade On-line
CONCRETE SERVICES

Figure 3: A portion of the service ontology for the running example.

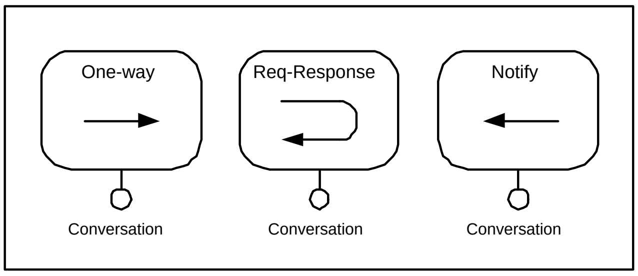


Figure 4: WebML primitives for Web services integration

and finally reducing the number of concrete services on the basis of context and quality requirements in an adaptive way.

4.1 Integrating Web Services and WebML

Concerning our scenario of the shipping company *ShipEveryWhere*, we require primitives capable of modeling interactions with external *Web Services*, due to the fact that the MAIS architecture supplies (abstract) Web services, which may be weaved into the application logic of a particular *User Environment*. [5] introduces the required functionalities at an adequate level of abstraction; Figure 4 shows the graphical rendition of the units used in our example. Appendix A shows the WebML constructs used for this purpose.

The depicted three operations represent just a subset of the introduced novel operations reflecting the set of WSDL message exchange patterns [13], but still enough for our purpose. The *One-way* operation serves the purpose of client-initiated messages, while the *Notify* operation stands for the inverse communication direction and thus for service-initiated messages. Finally, the *Request-Response* unit represents a synchronous operation initiated by users, with one outbound message followed by one inbound message. For further details about Web services integration into WebML please refer to [5, 6, 17].

4.1.1 Data Modeling

The first step in designing the user interface regarding the van driver consists of modeling the application data. Starting from the default WebML sub-schema, required for user management and personalization, three entities (*Van, Package, Route*) model the specific application data. The execution of service-related operations causes implicit update of data. In particular, the operations *GetPackageList* and *GetRoute/NewRoute* affect the entities *Package* and *Route* respectively.

4.1.2 Hypertext Modeling

Figure 6, finally, shows the arrangement of a possible hypertext built upon the specified data model and gives an idea of the complexity masking power of the *MAIS Platform*. Only operations exposed by means of *abstract MAIS services* are known at the *User Environment* level, while all the process orchestration details occur in a completely transparent manner. The names of used operations are referred to the *User Environment* and are directly mapped onto the operations of the process as described in Figure 1: incoming or outgoing links with respect to the box called *ShipEveryWhere*, which represents the overall delivery ser-

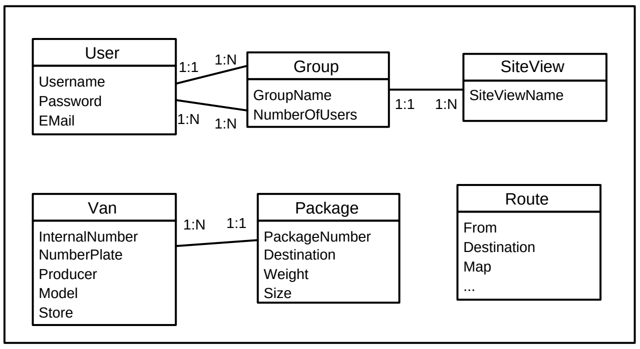


Figure 5: Data model for integrating the *ShipEveryWhere* service

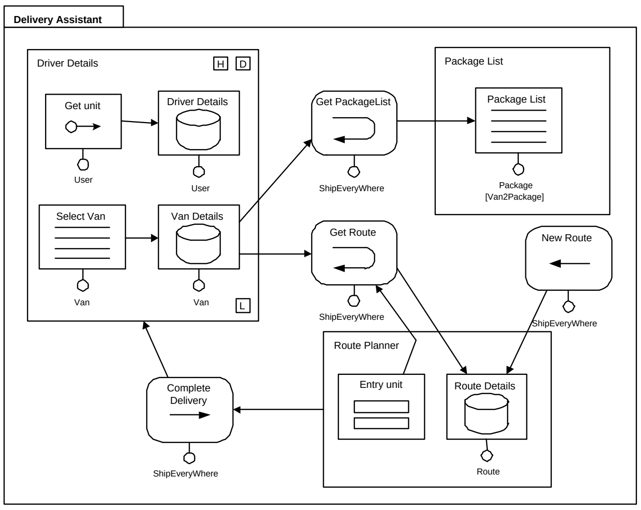


Figure 6: Hypertext schema of user interface.

vice, correspond to user-oriented invocations.

The hypertext schema describes the PDA user interface of the van driver. After a successful login process (not modeled here), the page *Driver Details* shows the relative user details and provides a list of free vans. The driver chooses one of the vans and, based on the chosen van, he can either get the list of the loaded packages or get the delivery route for the freight. Both these operations are performed by means of calls of the delivery Web service. While visiting the *Route Planner* page, the route can even change automatically due to changing traffic conditions and notified by means of the notify unit *NewRoute*. At the other hand, also the driver himself may send manually an *UpdateRoute* message and wait for the asynchronous answer *New Route*. Once the packages have been delivered, the *CompleteDelivery* operation communicates the successful delivery back to the control center.

5. RELATED WORK

The semantic description of services is very important in dynamic contexts where different services can offer, completely or partially, the requested features. The use of a registry that publishes and subscribes capabilities is the usual way to allow a dynamic search of services. The de-facto standard for registries is UDDI and nowadays all the semantic match-makers must be UDDI-compliant. The description of interfaces by means of WSDL is UDDI-compliant, but it is not enough to perform useful semantic search in a service registry. On the other hand, the use of rich descriptions, followed by the OWL-S coalition [1], can raise problems like the compliancy with UDDI and the delay associated with searches.

A compromise is described in [14, 25], where the authors propose a semantic description of services and a match-maker able to browse a UDDI-compliant registry. Our approach follows this compromise since the semantic description is used to improve the degree of freedom in the design of the business process, but search performances are still acceptable.

Another important aspect of service provisioning concerns the definition of languages for Quality of Service (QoS) descriptions. QoS has been the topic of several research and standardization efforts across different communities [21, 24, 26]. In [18], authors propose a multilayer model to evaluate quality of services in a dynamically evolving environment.

The adaptivity to the context is a fundamental issue of modern frameworks for provisioning of services. The adaptation process can involve or not the user. According to the degree of user interaction, we can identify three different levels.

At the lower level, adaptivity is focused on the middleware for service provisioning [8, 19]. In this perspective the nature of the application is weakly considered and often the user does not know or interact with the adaptation process.

The middle level is related to adaptivity issues on the business logic. Here, applications can react to events forwarded by the lower levels and modify their business logic in order to adapt their behavior with respect to users. Several systems and approaches have been proposed to extend traditional workflow management system technology to adaptive, Internet-based scenarios: CrossFlow [12] , WISE [16], MENTOR-LITE [22]. e-FLOW [9] is one of the first research prototypes addressing the issues of specifying, enacting and monitoring composite services; other proposals include SELFSERV [3], in which services can be composed and executed in a decentralized way, and The Dysco project [20] that faces the issue of automatic composition.

The top level involves aspects related to user environments [7]. Applications modify their user interfaces according to the client execution context. Automatic transcoding tools, like WebML [10], are very important in the automatic generation of multi-channel access systems.

6. CONCLUSIONS AND FUTURE WORK

In this paper we have presented a novel approach for the provisioning of complex abstract services. The decoupling of abstract description of services and their actual implementations is strongly exploited by the MAIS architecture and it was designed with this purpose in mind. Our service ontology is defined by looking a compromise between the richness of the description and its real usability. The definition of QoS dimensions become the fundamental parameter for the selection of the best service.

The possibility of dynamic search is already a kind of adaptivity. Moreover to increase the flexibility of our framework, we can provide simple services that have to be orchestrated by the end user or the architecture can hide all details and present only a value-added (fully orchestrated) service.

The adaptivity is also addressed by using a reflective architecture, which is able to know and, in some case manage, the parameters of the distribution channels.

Even if exiting languages give many opportunities, it is necessary to augment some of them. We are now formalizing these extended languages. The next step will be the implementation and deployment of the MAIS framework in some special-purpose settings.

Acknowledgments

This work is partially funded by the Italian MURST-FIRB MAIS Project (Multi-channel Adaptive Information Systems).

7. REFERENCES

[1] A. Ankolekar, M. Burstein, J. Hobbs, O. Lassila, D. Martinand D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, and K. Sycara. DAML-S: Web Service Description for the Semantic Web. In *In Proc. of International Semantic Web Conference (ISWC 2002)*, Chia, Italy, 2002.

[2] V. De Antonellis, M. Melchiori, B. Pernici, and P. Plebani. A methodology for e-Service substitutability in a virtual district environment. In *Proc. of 15th International Conference on Advanced Information Systems Engineering (CAiSE 2003)*, volume 2681 of *Lecture Notes in Computer Science*, pages 552–567, Klagenfurt, Austria, June 16th-20th 2003. Springer.

[3] B. Benatallah, Q. Sheng, and M. Dumas. The Self-Serv Environment for Web Services Composition. *IEEE Internet Computing*, 7(1):40–48, 2003.

[4] D. Bianchini, V. De Antonellis, and M. Melchiori. An ontology-based method for classifying and searching e-Services. In *Proc. Forum of First Int. Conf. on Service Oriented Computing (ICSOC 2003)*, Trento, Italy, December 15th-18th 2003.

[5] M. Brambilla, S. Ceri, S. Comai, P. Fraternali, and I. Manolescu. Model-driven Specification of Web Services Composition and Integration with Data-intensive Web Applications. *Bulletin of the Technical Committee on Data Engineering*, 25(4), December 2002.

[6] M. Brambilla, S. Ceri, S. Comai, P. Fraternali, and I. Manolescu. Model-driven development of web services and hypertext applications, December 2003. SCI2003, Orlando, Florida.

[7] P. Brusilovky. Adaptive hypermedia. *User Modeling and User Adapted Interaction*, 11(1-2):87–100, 2001.

[8] L. Capra, W. Emmerich, and C. Mascolo. CARISMA: Context-Aware Reflective middleware system for Mobile Applications. *IEEE Transactions on Software Engineering*, 29(10):929–945, 2003.

[9] F. Casati and M. Shan. Dynamic and Adaptive Composition of e-Services. *Information Systems*, 26(3):143–163, May 2001.

[10] S. Ceri, P. Fraternali, B. Bongio, M. Brambilla, S. Comai, and M. Matera. *Designing Data-Intensive Web Applications*. Morgan Kauffmann, 2002.

[11] F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte, and S. Weerawarana. *Business*

Process Execution Language for Web Services, version 1.0, July 2002.

[12] P. Grefen, K. Aberer, Y. Hoffner, and H. Ludwig. CrossFlow: Cross-Organizational Workflow Management in Dynamic Virtual Enterprises. *International. Journal of Computer Systems Science & Engineering*, 15(5):277–290, 2000.

[13] Martin Gudgin, Amy Lewis, and Jeffrey Schlimmer. Web Services Description Language (WSDL) Version 2.0 Part 2: Predefined Extension, August, 3rd 2004.

[14] T. Kawamura, J.A. De Blasio, T. Hasegawa, M. Paolucci, and K. Sycara. Preliminary Report of Public Experiment of Semantic Service Matchmaker with UDDI Business Registry. In *Proc. of First Int. Conf. on Service Oriented Computing (ICSOC 2003)*, volume 2910, pages 208–224, Trento, Italy, December 15th-18th 2003. Lecture Notes in Computer Science Springer-Verlag.

[15] J. Krogstie, K. Lyytinen, A. L. Opdahl, B. Pernici, K. Siau, and K. Smolander. Research areas and challenges for mobile information systems. *International Journal of Mobile Communication (IJMC). Special issue on Modeling Mobile Information Systems: Conceptual and Methodological Issues*, 2(3), 2004.

[16] A. Lazcano, G. Alonso, H. Schuldt, and C. Schuler. The WISE approach to Electronic Commerce. *International Journal of Computer Systems Science & Engineering*, 15(5), September 2000.

[17] I. Manolescu, S. Ceri, M. Brambilla, P. Fraternali, and S. Comai. Exploring the combined potential of web sites and web services. poster at WWW03, Budapest, Hungary.

[18] C. Marchetti, B. Pernici, and P. Plebani. A Quality Model for Multichannel Adaptive Information Systems. In *Alternate Tracks Proceedings of 13th International World Wide Web Conference (WWW2004), ACM Press*, pages 48–54, New York City, NY, USA, May 17th-22th 2004.

[19] N. Parlavantzas, G. Coulson, and G.S. Blair. A Resource Adaptation Framework For Reflective Middleware. In *Proc. of 2nd Int. Workshop on Reflective and Adaptive Middleware (located with ACM/IFIP/USENIX Middleware 2003)*, pages 163–168, Rio de Janeiro, Brazil, June 2003.

[20] G. Piccinelli and L. Mokrushin. Dynamic e-Service composition in DySCo. In *In Proc. of Int. Workshop on Distributed Dynamic Multiservice Architecture, at ICDCS*, Phoenix, Arizona, USA, 2001.

[21] Shuping Ran. A Model for Web Services Discovery with QoS. In *ACM SIGecom Exchange*, volume 4, pages 1–10, ACM Press, New York, NY, USA, 2003.

[22] G. Shegalov, M. Gillmann, and G. Weikum. XML-enabled Workflow Management for e-Services across Heterogeneous Platforms. *VLDB Journal: Very Large Data Bases*, 10(1):91–103, 2001.

[23] The MAIS Project Team. The MAIS Project. In *Proc. of 4th International Conf. on Web Information Systems Engineering*, Rome, Italy, December 2004.

[24] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Z. Sheng. Quality driven web services composition. In *In Proc. of Conference on World Wide Web*, pages 411–421. ACM Press, 2003.

[25] L. Zeng, B. Benatallah, Anne H.H. Ngu, Marlon Dumas, Jayant Kalagnanam, and Henry Chang. QoS-Aware middleware for web services composition. *IEEE Trans. on Software Engineering*, 30(5):311–327, May 2004.

[26] John A. Zinky, David E. Bakken, and Richard E. Schantz. Architectural support for quality of service for CORBA objects. *Theory and Practice of Object Systems*, 3(1):1–20, 1997.

APPENDIX

A. THE WEB MODELING LANGUAGE

WebML is widely known for being an intuitive visual language for specifying the structure of data-intensive Web applications and the organization of contents in one or more hypertexts [10]. However, in a certain sense, it is even more than yet another specification language. Indeed, it can also be considered a full design process consisting of two main activities, which represent incremental steps towards the final application:

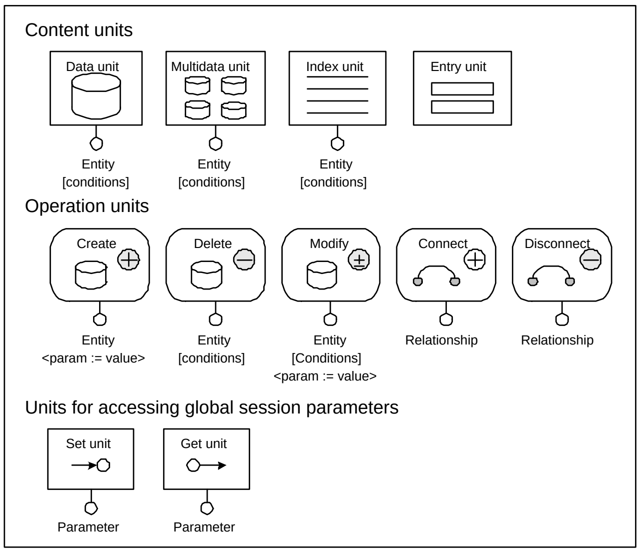


Figure 7: Summary of core WebML units.

- **Data Design**. The WebML *Data Model* represents the basis for the overall modeling process and adopts the Entity-Relationship (ER) primitives for representing the organization of the application data. Its fundamental elements are therefore entities, attributes and relationships.
- **Hypertext Design**. The WebML *Hypertext Model* allows describing how contents, specified by means of the ER data schema, are published into the application hypertext, the so-called *site views*. Site views are structured by *areas* and *pages*, that are the actual content containers made of *content units*. They are directly associated with data entities and, by means of specific selector conditions, publish content within pages. Besides content units, *operation units* provide support for content management operations, *set* and *get units* allow accessing session variables and *entry units* model

HTML input forms. Units and pages are interconnected by *links*, transporting or not parameters and describing user navigation. Figure 7 shows a graphical summary of core WebML units.

Personalization of contents and services is achieved by modeling users and their roles as data. Personalization may occur along two different dimensions: customized contents with respect to user identity and tailored hypertext structure with respect to groups the user belongs to (e.g., guest, adiministrator and so on). The first is based on relationships between users and content entities at data level, the latter requires designing alternative site views for each user group.

Site views may also serve the purpose of expressing alternative forms of content organizations on different devices for the purpose of *multi-channel deployment*. Each site view may cluster information and services at the granularity most suitable to a particular class of devices or communication protocol.

Yet WebML does not provide any delivery mechanism, nor does it depend on the particular deployment language chosen for application delivery. Its visual representation, though, is mapped on an equivalent XML-based textual representation that can be processed by *automatic code generation* tools, such as the *WebRatio Site Development Studio*.

Web Services Orchestration and Interaction Patterns: an Aspect-Oriented Approach

Guadalupe Ortiz Juan Hernández Pedro J. Clemente

Quercus Software Engineering Group
University of Extremadura
Computer Science Department
Avda. de la Universidad s/n. 10071 Cáceres, Spain

+ 34 924 38 70 68 + 34 927 25 72 04 + 34 927 25 78 07

gobellot@unex.es juanher@unex.es jclemente@unex.es

ABSTRACT
Web Service technologies offer a new and successful way for interoperability among web applications. However, there is not a unique and standard opinion as to how Web Services composition must be implemented, and services involved are generally strongly coupled, which raises problems at design, implementation, maintenance and evolution**.** This paper shows one approach to the implementation of orchestrations by using aspect-oriented techniques, thus improving modularity, scalability and flexibility in the compositions. Aspect-oriented programming will also allow us to reuse the interaction patterns described by the orchestrations in different contexts, as we will demonstrate in this research.

Categories and Subject Descriptors
D.3.3 **[Programming Languages]**: Language Constructs and Features – *abstract data types, classes and objects, patterns.*

D.1.0 **[Programming Techniques]**: General.

General Terms
Design, Languages.

Keywords
Web services compositions, orchestrations, business processes, aspect-oriented programming.

1. INTRODUCTION
Web Services convey one step further in the long way which object-oriented technologies and distributed platforms have walked. These technologies offer a new and successful solution for interoperability among web applications, and they have become the best way to integrate third-party approaches, therefore collaborating in the client-server architectures replacement by peer-to-peer distributed architectures [4].

Once the general behaviour and definition of Web Services seem highly consolidated, it is time to face how to tackle interaction among different services. Unfortunately, there is so far no agreement on how to implement Web Services composition. Whereas different proprietary approaches rise in the business process control, and various standards try to emerge in order to solve future business connectivity, there is not yet any free approach to compose Web Services in an easy way.

The terms *orchestration* and *choreography* [11] have recently emerged to be two of the biggest attention points on Web Services nowadays. They refer to two different ways for managing business connectivity, and have arisen in a moment in which many companies have begun to incorporate Web Services to their deployments. Many languages have been proposed and discussed for those types of collaborations among business processes, for instance, XLANG [13], WSFL [8], or BPML [2], but we can especially mention three of them in the area of Web Services: BPEL4WS [1], WSCI [3] and WS-CDL [6]. Whereas the first three are more oriented to business and flow control, the last above mentioned are especially led to Web Services and their composition, and this is precisely what we are going to focus on in this paper.

BPEL4WS (*Business Process Execution Language for Web Services*) allows users to describe the control logic for coordinating different Web Services which takes place in a process flow, that is, the way in which the invocations may be ordered. It is mainly focused on permitting orchestration to be defined, although abstract BPEL4WS attempts to describe external observable behaviour of single services to be used for choreography descriptions.

On the contrary, WSCI (*Web Service Choreography Interface*) is based on the particular description of each service and the way in which they all are choreographed, and it only describes the observable interaction of Web Services with their users, which may also be a Web Service. WSCI was proposed by Intalio, and it was one of the working notes in the development of the W3C Working Group. It is clearly oriented to choreographies, not to orchestrations, but it does not provide a good notion of the global model in the interactions.

WSCDL (*Web Service Choreography Description Language*) is yet one more proposal, from Oracle this time. As its name indicates, it is also oriented to choreographies, and the W3C workgroup has recently published a draft of it. So we would expect a standard on choreographies in the same line as Intalio and Oracle's proposals, which is currently being elaborated by the W3C workgroup.

Regarding orchestration, BPEL4Ws appears to be the most relevant approach, although, as we mentioned before, the boundaries between choreographies and orchestrations are not very clearly defined and therefore we cannot establish limits concerning them for the different approaches proposed.

In any case, what is evident is that we still have no established standard for composing Web Services and that the approaches which are already in the market are proprietary, complex and rather highly business logic-oriented, neither do they offer the possibility of reusing interaction patterns previously implemented as compositions. What happens if we have already defined an orchestration and we want to reuse the same interaction pattern in another one? Do we have to code all the composition again? Could we not have the pattern in a modularized way instead of having all the code scattered and highly coupled to the main application? In spite of it being influential to both evolution and maintenance in Web Services, the market has not found an answer to this matter for the time being.

For these reasons, we propose composing Web Services by using *Aspect-Oriented Programming* (AOP) [7], thus totally decoupling the various Web Services composed and facilitating services maintenance and modularization, as well as reusability of their interaction patterns. In this paper we shall centre mainly in how to apply AOP techniques to Web Services composition and how to reuse the interaction logic of the orchestration previously defined.

The rest of the paper will be organized as follows: a case study is presented in section 2 to identify the problems defined before, illustrating the orchestration concept and its implementation using different kinds of tools. Section 3 outlines the way in which AOP can help to solve these problems, and how AspectJ has been used and applied in Web Services composition, as well as in defining and implementing orchestrations interaction patterns; in this sense, we have illustrated the matter with some code examples. Other related approaches are discussed in section 4, and the main conclusions will be presented in section 5.

2. WEB SERVICES ORCHESTRATION

The term *orchestration* [11] has recently emerged to be one of the biggest attention points concerning Web Services nowadays. It is related to the way in which business connectivity is managed and has appeared at a moment in time in which many companies had begun to incorporate Web Services to their deployments.

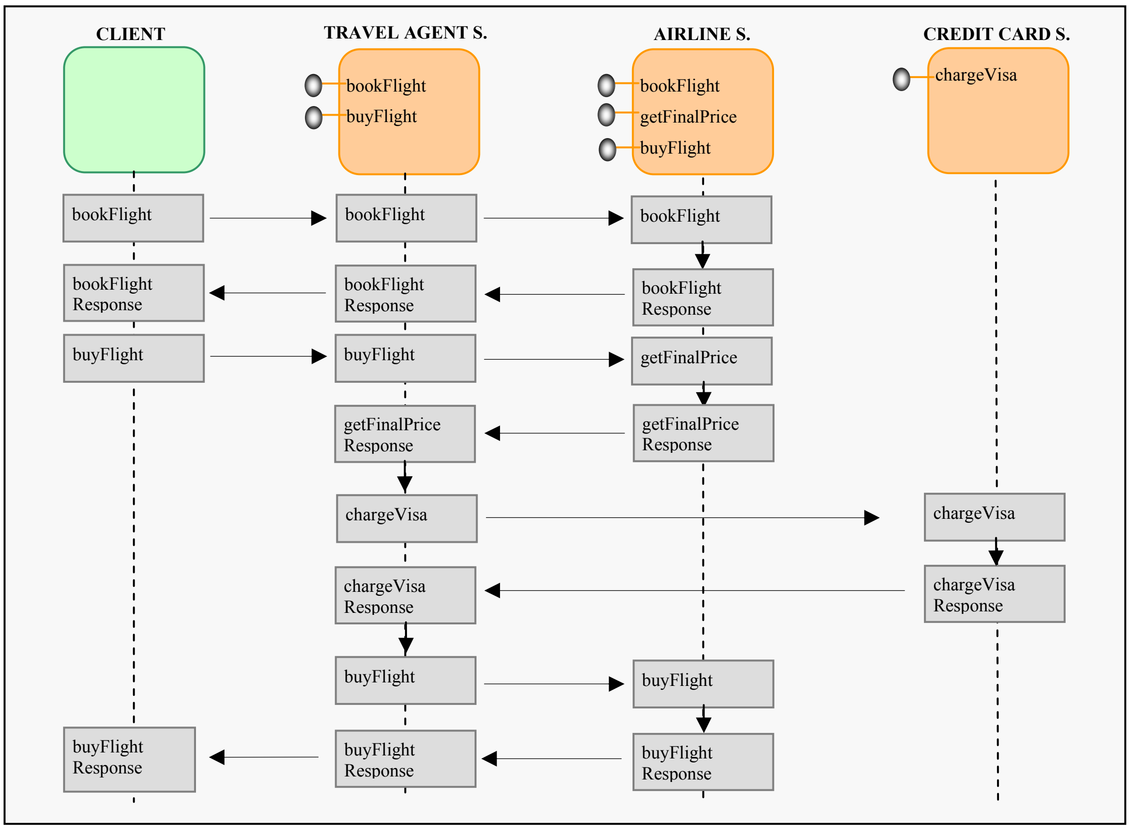


Figure 1. Invocation order in an orchestration example.

Orchestrations refer to the sequence of activities that compose a business process, in other words, to Web services which interact among them and that probably may imply an execution order in the messages invocations, as well as a business logic. Orchestration includes the management of the different messages interchanged among the services involved, and it is important to note that one of the parties controls the process.

2.1 Case Study

For instance, as shown in Figure 1, we may have an orchestration composed of three services: a travel agent service, an airline service and a credit card service. To begin with, the travel agent, the part which controls the process, offers two operations: *String bookFlight (int passengerNumber, date flightDay, String flightNumber)* and *String buyFlight (String reservationNumber, int creditCardNumber)*, which allow him to book and buy tickets for a specific flight, respectively. Secondly, the airline service offers the same two operations, but the travel agency identification must be provided instead of the credit card number of the client: *String bookFlight (int passengerNumber, date flightDay, String flightNumber)* and *String buyFlight (String reservationNumber, int agencyID).* It also offers one additional operation for obtaining the final price of the selected flight: *int getFinalPrice(String flightNumber).* Finally, the credit card service offers a single operation, *String chargeVisa (int creditCardNumber, int amount)*, in order to charge the cost of the flight to the client's credit card. We can observe the control flow in the above figure, in which a client has been included.

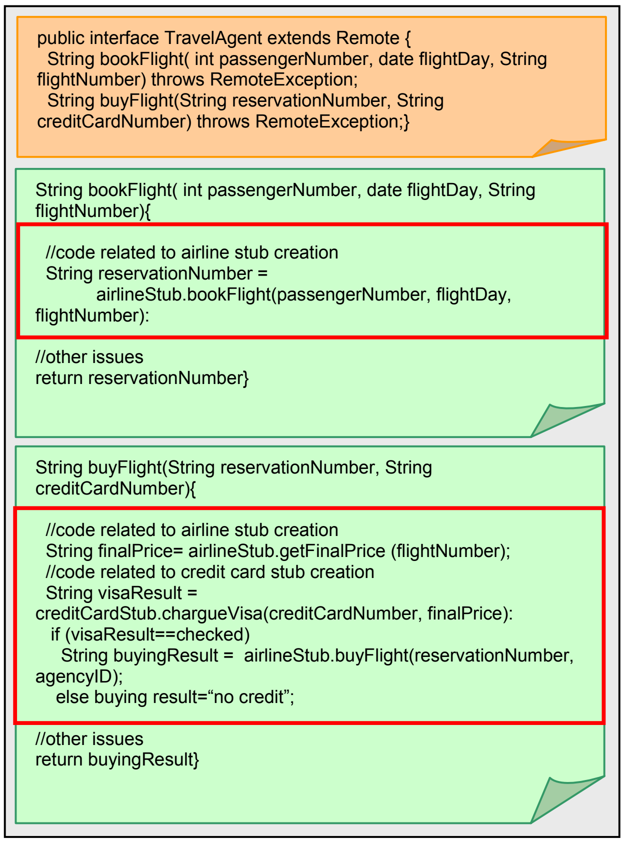
The client asks the travel agent to book a flight, and, automatically, the travel agent asks the same of the airline service. When the airline provides the reservation number, if booking is possible, the agent in turn gives it to the client. Likewise, when the client requests the flight purchase, the travel agent first of all has to ask the airline for the final price of the flight; afterwards he has to charge it to the client's credit card and finally, if everything is right, the travel agent buys the ticket from the airline service. Once the process is finished, it returns to the client, whatever the result of the operation is.

2.2 Orchestration Implementations

In this section we are going to implement the orchestration defined in Figure 1 with two different tools, exemplifying the two main approaches for Web Service compositions. The first one, the Sun tool JWSDP, is based simply on the implementation of code with a general tool for Web Services performance; the second one, BPEL4WS, is based in the creation of a new XML file that leads the orchestration.

2.2.1 Web Service Orchestration Using JWSDP

To begin with, if we implement the orchestration example using free conventional tools for Web Services, we would have to implement all the code related to the composition manually. In the case of the travel agent service the orchestrations would be implemented in both operations, *bookFlight* and *buyFlight*, as shown in Figure 2. As can be seen in the figure mentioned, this implementation does not perform many of the basic mainstay of programming, as we will briefly go on to outline:



```
public interface TravelAgent extends Remote {
  String bookFlight( int passengerNumber, date flightDay, String
flightNumber) throws RemoteException;
  String buyFlight(String reservationNumber, String
creditCardNumber) throws RemoteException;}
```

```
String bookFlight( int passengerNumber, date flightDay, String
flightNumber){

  //code related to airline stub creation
  String reservationNumber =
          airlineStub.bookFlight(passengerNumber, flightDay,
flightNumber):

//other issues
return reservationNumber}
```

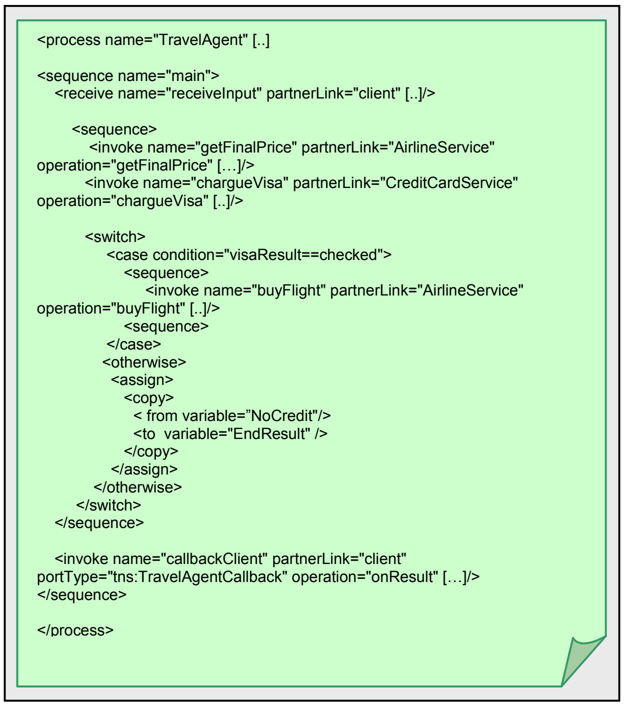
```
String buyFlight(String reservationNumber, String
creditCardNumber){

  //code related to airline stub creation
  String finalPrice= airlineStub.getFinalPrice (flightNumber);
  //code related to credit card stub creation
  String visaResult =
creditCardStub.chargueVisa(creditCardNumber, finalPrice):
   if (visaResult==checked)
     String buyingResult =  airlineStub.buyFlight(reservationNumber,
agencyID);
    else buying result="no credit";

//other issues
return buyingResult}
```

Figure 2. Travel Agent orchestrations with JWSDP.

- Modularity: The code related to the other Web Services invocation is very strongly coupled, which creates hard dependencies between the main service and the rest of the services which are involved in the composition, thus having all the code related to the orchestration mixed and scattered in the application and therefore definitely not modularized.
- Reusability: If we want to reuse the same orchestration we have no chance to do so, as all the code is tangled and scattered all over our application. Furthermore, we cannot reuse the interaction pattern in other contexts where we could find it, either.
- Maintenance and evolution: In all the operations in which we need to interact with other services, such as the airline service, we have to create a local stub in order to invoke its operations, which implies not only a bad design, but also problems in maintenance and evolution. Furthermore, if we want to offer a new operation in our travel agent in which the airline service is also involved, we will have to repeat the same code again.

2.2.2 Web Service Orchestration Using BPEL4WS

On the other hand, if we try to implement the same example with the BPEL4WS tool, we find it is a visual tool that generates an XML code for the indicated interactions. The code related to the implementation of the orchestration for the *buyFlight* operation can be seen in Figure 3. It has been simplified to emphasize the interaction sequence code.



```
<process name="TravelAgent" [..]

<sequence name="main">
   <receive name="receiveInput" partnerLink="client" [..]/>

      <sequence>
         <invoke name="getFinalPrice" partnerLink="AirlineService"
operation="getFinalPrice" […]/>
        <invoke name="chargueVisa" partnerLink="CreditCardService"
operation="chargueVisa" [..]/>

        <switch>
           <case condition="visaResult==checked">
              <sequence>
                 <invoke name="buyFlight" partnerLink="AirlineService"
operation="buyFlight" [..]/>
              <sequence>
           </case>
          <otherwise>
           <assign>
              <copy>
                < from variable="NoCredit"/>
                <to  variable="EndResult" />
              </copy>
           </assign>
         </otherwise>
      </switch>
   </sequence>

   <invoke name="callbackClient" partnerLink="client"
portType="tns:TravelAgentCallback" operation="onResult" […]/>
</sequence>

</process>
```

Figure 3. *BuyFlight* orchestration with BPEL4WS.

If we examine the same points in this case as we did in the previous subsection with the JWSDP implementation, we will find that we have improved as far as modularity, maintenance and evolution are concerned but we again find an essential not desirable aspect for an application:

- Reusability: In this case, we can reuse the orchestration completely, as a new service, provided we have exactly the same requirements in another application, in which case we would invoke the operations offered by the orchestration. However, we still have no chance to reuse the interaction pattern in other contexts. Since it is a very common pattern, it would be very useful to have a unique interaction pattern to specialize it to the different contexts where we want to apply it.

Therefore, these are the reasons why we have to look for an easier way to compose services, which will allow us to maintain our services independent and our final application well modularized and structured, avoiding the composition code being scattered and tangled throughout the application and thus facilitating maintenance and evolution in our orchestrations.

Although BPEL could be a solution to avoiding scattered and mixed code in the main application, the reusability of the interaction pattern is also very important, as we mentioned before, since we can find many contexts in which the same pattern may appear. For example, we can find examples as a portal for selling cinema tickets or any other portal for selling products over the Internet which may easily adapt to the pattern in our example. Therefore, it is very desirable to be able to reuse and adapt the pattern to our particular example, instead of needing to create the interaction flow again.

We believe AOP is the answer to the problem, since it was created in order to deal with elements that are scattered all over an implementation, by modelling their behaviour in an external layer. Aspect-oriented techniques are also successfully used in the Web Service domain for decoupling non-functional properties at compilation time, allowing the implementation of important properties in the Web Services, as logging or timing, in a modularized and completely decoupled way [9]. Hence, it will probably be helpful in decoupling these compositions and allowing to reuse their interaction patterns. As a result, we would be able to manipulate the orchestrations without influencing the rest of the code in the application, having the code related to the remote services involved in the composition completely modularized and being able to reuse their interaction patterns should they be necessary.

3. ORCHESTRATIONS IMPLEMENTATION AND REUSE WITH AOP

In this section we are going to show how aspect-oriented techniques may be used in order to solve the above difficulties, that is, to avoid the strong dependencies among the composed services and to allow reuse of their interaction patterns.

AOP arises because of the problems detected in *Object-Oriented Programming* (OOP). OOP is supposed to permit the encapsulation and modularity of related data and methods which address a common goal. This should imply a code completely organized in meaningful units and not blended at all, but this is not always possible. We may find it impossible to model various concerns into a unique and structured decomposition of units. We can simply have transversal concerns, which cannot be included in the logical structuring of the code by functionality. These concerns cause scattering and tangling code all over our application and that is the reason why they are called crosscutting concerns.

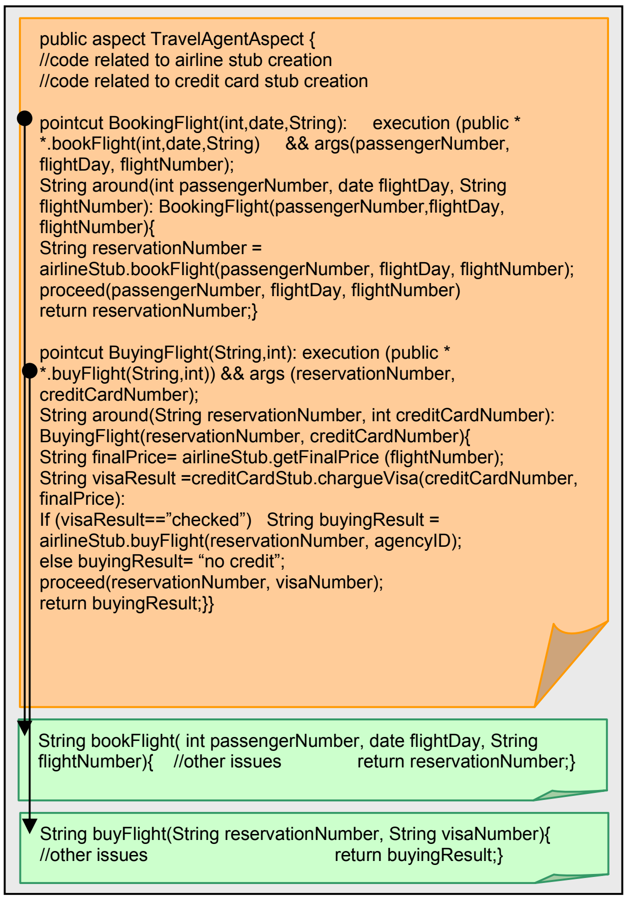
AOP establishes *aspect* as the way to model these crosscutting concerns. *Aspects* are units of encapsulation which incorporate two main elements: *join points* and *advices*. On the one hand, through *join points* we specify in which points of the implementation we wish to insert the new code, that is, where we want to alter the behaviour in the application. On the other hand, *advices* identify the new code to be injected, thus reflecting the desired new behaviour in the application.

In the orchestration presented in Figure 1, which offers us the interaction flow of the compositions, we can see how we have to follow an order when making the invocations: we will then have services which have to be invoked before others, those which will have to be invoked after others have finished and finally services which may be or may not be invoked depending on the result of other invocations. Aspect-oriented programming allows us to model this kind of interactions in separated aspects, that is, in individual units. Furthermore, all the business rules remain modelled in the aspect, without having scattered code due to the composition in the application.

We can use several different languages to model aspects. We have chosen AspectJ among all aspect-oriented languages, both because of its proximity to Java programming language, and because it is very versatile, offering plenty of possibilities at design and implementation time.

3.1. Implementing Orchestrations

Figure 1 indicated the interaction flow for implementing the travel agent orchestrations, the interaction flow for booking a flight and the one for buying it. Aspect-oriented languages as AspectJ model this kind of interactions into different independent units called *aspects*. This can be seen in Figure 4, where we can notice how the aspect implements the interaction pattern for the travel agent operations. In this figure we can discern *pointcut BookingFlight()*, which injects code in execution of method *bookFlight*. The corresponding *advice* (an *around advice*) indicates the code to be injected. In this same figure we can recognize *pointcut BuyingFlight()*,which injects code in execution of method *buyFlight*. The corresponding *advice* (an *around advice*) shows the code to be injected.



```
public aspect TravelAgentAspect {
//code related to airline stub creation
//code related to credit card stub creation

pointcut BookingFlight(int,date,String):     execution (public *
*.bookFlight(int,date,String)     && args(passengerNumber,
flightDay, flightNumber);
String around(int passengerNumber, date flightDay, String
flightNumber): BookingFlight(passengerNumber,flightDay,
flightNumber){
String reservationNumber =
airlineStub.bookFlight(passengerNumber, flightDay, flightNumber);
proceed(passengerNumber, flightDay, flightNumber)
return reservationNumber;}

pointcut BuyingFlight(String,int): execution (public *
*.buyFlight(String,int)) && args (reservationNumber,
creditCardNumber);
String around(String reservationNumber, int creditCardNumber):
BuyingFlight(reservationNumber, creditCardNumber){
String finalPrice= airlineStub.getFinalPrice (flightNumber);
String visaResult =creditCardStub.chargueVisa(creditCardNumber,
finalPrice):
If (visaResult=="checked")   String buyingResult =
airlineStub.buyFlight(reservationNumber, agencyID);
else buyingResult= "no credit";
proceed(reservationNumber, visaNumber);
return buyingResult;}}

String bookFlight( int passengerNumber, date flightDay, String
flightNumber){    //other issues             return reservationNumber;}

String buyFlight(String reservationNumber, String visaNumber){
//other issues                              return buyingResult;}
```

Figure 4. Travel Agent orchestrations using aspects.

Besides, all the business control remains modelled in the same unique aspect, without having code scattered and tangled all over the application. We can note the fact that there is no reference to the specific airline service to invoke in the application code in Figure 4, in contrast with the original code, without aspects, previously shown in Figure 2. As can also be seen in this code, all the order in the invocations is controlled by the advice, not by the corresponding operation, thus abstracting, modularizing and encapsulating the code related to the rest of the services involved in the composition. Furthermore, we do not have repeated or scattered code because of the different stub instances for the different modules in the application, as we have all modularized in the aspect code.

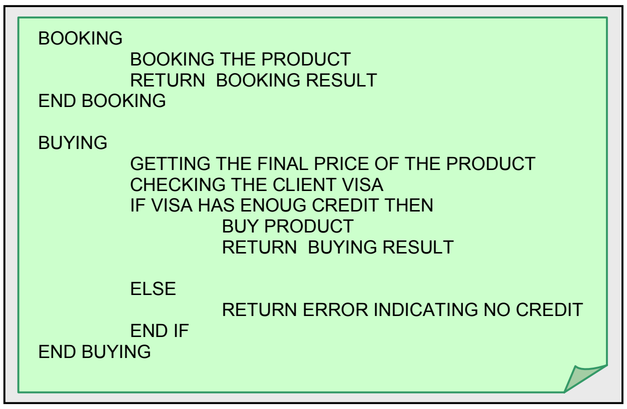
The JWSDP tool leans on the Ant tool for compilation processing, thus for compiling this AspectJ code with the rest of the application, we will have to modify the *build.xml* file. We must link the AspectJ compiler, instead of the Java one, in the compilation targets inside the *build* file.

Therefore, aspect orientation may be used to model control flow in Web Services orchestrations, preserving the participating services decoupled and independent among themselves. Hence, if we want to replace any of the services invoked from the travel agent by another one, we can do it without modifying the main application, but by only altering the aspect class.

Hence, our proposal allows the developer to implement the applications, without minding which services have to be invoked from it, and focusing on the main structure and the essence of the application. After that, he can attach the needed invocations via aspects, thus maintaining the code well modularized and encapsulated. In the very moment the developer decides to replace one of the services invoked by another one due to new requirements, the main application will not be affected and he will only have to modify the aspect. In this regard and without affecting the main code behaviour or structure, we could add non-functional properties to the services involved in the composition or to the orchestration itself by using aspects. For example, we could add a timing property to be able to check the time of use of the different services involved or the time of use of the different operations offered by the orchestration. The same could be done with the logging property [9].

3.2. Reusing the orchestrations interaction patterns

We stressed in the previous section how the impossibility of reusing the interaction pattern was one of the most important drawbacks of implementing orchestrations not only with conventional tools, but also with specific tools as BPEL4WS. The interaction pattern of the case study is represented in Figure 5.



```
BOOKING
        BOOKING THE PRODUCT
        RETURN  BOOKING RESULT
END BOOKING

BUYING
        GETTING THE FINAL PRICE OF THE PRODUCT
        CHECKING THE CLIENT VISA
        IF VISA HAS ENOUG CREDIT THEN
                BUY PRODUCT
                RETURN  BUYING RESULT

        ELSE
                RETURN ERROR INDICATING NO CREDIT
        END IF
END BUYING
```

Figure 5. Interaction pattern for booking and buying products over the Internet.

This pattern is very usual nowadays. Many companies offer the possibility of buying some products, previously booked online, checking before the end of the purchase whether the credit card provided by the client has enough credit. Most the electronic commerce applications could adapt to this pattern. In this sense we can create a pool of typical patterns implemented as a composition of services, as for instance a purchase with a previous credit and stock check or related pursuits based on the result of previous searches…, so we can reuse the patterns in the different contexts in which they appear.

By using aspect-oriented techniques, the interaction patterns can be implemented as abstract aspects so we can also obtain a big capacity of reuse from using AOP for performing compositions. We may define an abstract aspect which describes the interaction pattern for a specific kind of orchestration, as we have depicted in Figure 6, which implements the interaction pattern of the case study presented.

```
abstract aspect AbstractCompositionAspect {

public abstract String booking(Object [] args);
public abstract String gettingPrice (Object [] args);
public abstract String checkingBalance (Object [] args, String
finalPrice);
public abstract String buying (Object [] args);

abstract pointcut Booking ( );
String around( ): Booking (){
   Object[ ] args = thisJoinPoint.getArgs( );
   String reservationNumber =  booking(args);
   proceed (args[0], args[1]. args[2]);
   return reservationNumber;}

abstract pointcut Buying ( );
String around(): Buying ( ){
   Object[ ] args = thisJoinPoint.getArgs( );
   String finalPrice=gettingPrice(args);
   String visaResult =checkingBalance(args, finalPrice);
   if (visaResult==checked)
   String buyingResult = buying (args);
   else buyingResult="no credit";
   proceed(args[0], args[1]);
   return buyingResult;}}
```

Figure 6. Abstract aspect for a booking/buying orchestration.

Therefore, in the very moment we need to implement an orchestration of that kind, we only have to inherit the composition pattern from the abstract aspect and particularize it with the specific stubs from the services invoked and the specific invocation methods. As we can see, this aspect is integrated by two abstract *pointcuts*, one for booking and the other one for the purchase. As they are abstract aspects, we still do not have to specify in which method execution we are going to insert the new code. In the *advices* linked to those *pointcuts*, we can see how the arguments are obtained from the context and used for the invocation of the abstract classes that represent the operations of the remote methods. Regarding the booking of the product, we can stress that we only need to invoke the abstract method *booking*; and for the buying of the product we can point out that we first have to invoke the method *gettingPrice*, *checkingBalance* after that and, finally, if the credit card result is right, the final method *buying*; all of them abstract. We do not need to indicate the specific Web Services that have to be invoked nor their operations, since this will be done when we create the specific aspect that inherits the abstract one. Hence, we can represent the interaction pattern as an AspectJ aspect.

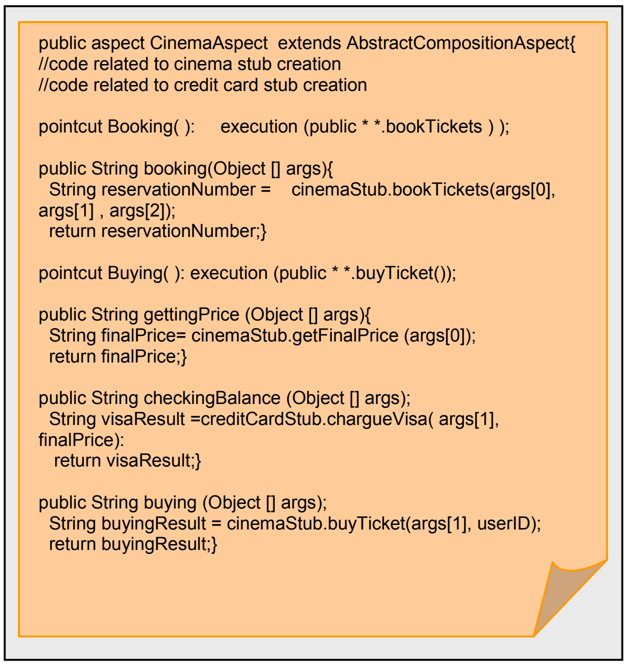
If we want to implement our specific orchestration case study using this designed pattern, we have to implement an aspect which extends the abstract one, specifying the concrete stubs of the services involved in the composition (*creditCardService, airlineService*), as can be seen in Figure 7.

```
public aspect CompositionAspect  extends
AbstractCompositionAspect{
//code related to airline stub creation
//code related to credit card stub creation

pointcut Booking( ):    execution (public * *.bookingFlight( ) );

public String booking(Object [] args){
  String reservationNumber =    airlineStub.bookFlight(args[0],
args[1] , args[2]);
  return reservationNumber;}

pointcut Buying( ): execution (public * *.buyFlight());

public String gettingPrice (Object [] args){
  String finalPrice= airlineStub.getFinalPrice (args[0]);
  return finalPrice;}

public String checkingBalance (Object [] args, String finalPrice);
  String visaResult =creditCardStub.chargueVisa(args[1],
finalPrice):
   return visaResult;}

public String buying (Object [] args);
  String buyingResult = airlineStub.buyFlight(args[1], agencyID);
  return buyingResult;}
```

Figure 7. Inherited aspect for a booking/buying orchestration.

We also have to redefine the *pointcuts Booking* and *Buying*, indicating the points in which we are going to insert the new code. After that we also have to redefine the abstract operations, *gettingPrice, checkingBalance* and *buying*, now to implement their real behaviour, invoking the corresponding operation in the remote services.

If now, for instance we wanted to make an orchestration to offer a web portal for buying cinema tickets online, we could reuse the same interaction pattern and make our new composition aspect inherit from it so as to offer the desired behaviour. As shown in Figure 8, the *pointcuts* are now extended with the methods of the cinema application, as well as the abstract methods include the invocation to the new Web Services implicated in the orchestration.



```
public aspect CinemaAspect  extends AbstractCompositionAspect{
//code related to cinema stub creation
//code related to credit card stub creation

pointcut Booking( ):    execution (public * *.bookTickets ) );

public String booking(Object [] args){
  String reservationNumber =    cinemaStub.bookTickets(args[0],
args[1] , args[2]);
  return reservationNumber;}

pointcut Buying( ): execution (public * *.buyTicket());

public String gettingPrice (Object [] args){
  String finalPrice= cinemaStub.getFinalPrice (args[0]);
  return finalPrice;}

public String checkingBalance (Object [] args);
  String visaResult =creditCardStub.chargueVisa( args[1],
finalPrice):
   return visaResult;}

public String buying (Object [] args);
  String buyingResult = cinemaStub.buyTicket(args[1], userID);
  return buyingResult;}
```

Figure 8. Inherited aspect for booking/buying tickets online for a cinema.

Once we have seen how to implement orchestration by using aspect-oriented techniques, how to perform their interaction patterns and how to reuse them, we can review the three programming pillars we mentioned before in order to check how we have improved the orchestrations implementation through the use of aspect-oriented programming:

- Modularity: As we have coded all the code related to the composition in the aspect, it is found completely modularized, decoupling the remote services entirely from the application we are implementing and thus having no trace of the scattered and tangled code that we had before using aspect-oriented programming techniques.
- Reusability: Now we can not only reuse the complete orchestration if we need exactly the same behaviour for another application, as we have it well modularized; we can also reuse interaction patterns previously defined, extending them in an specific aspect which implements the new orchestration.
- Maintenance and evolution: Now that we have the composition code separated, maintenance and evolution are better, as we know we do not have to modify the main code of the application at all for changing the interaction flow, but only the modularized aspect. In this sense, our application will be more reliable.

Regarding survivability, as the composition is specified in the aspect itself, it will survive as long as the signatures of the methods in the main application do not change, independently of whether the behaviour does. If method signatures changed, we would only have to modify the *pointcuts* defined in the aspects, maintaining the implemented behaviour. On the other hand, due to the big facilities existing concerning the use of aspects in different environments and the different aspect-oriented languages available we can also mark the adaptability of the proposal, not only to the environments, but also to changes as explained in the point of evolution.

Definitely, we can assert that AOP turns to be very useful for Web Service composition in general. Both in the case of orchestrations or just a simple invocation between two services, aspect-oriented techniques provide a good way to modularize and encapsulate the message exchange in whichever of the previous occurrences, decoupling the services among them and offering an easy maintenance of the application. Moreover, we can also define interaction patterns with AOP for orchestrations and reuse them should they be necessary.

4. RELATED WORK

Web Services composition is a very common research area nowadays. There are plenty of studies on Web Services composition and on how to improve them. Although there are undoubtedly important infrastructural issues in this field, there seems to be little or no discussion in the specialized media on how to use AOP techniques for this functionality.

On the one hand, the idea of encapsulating the composition logic and maintaining it in a modularized and decoupled form can be examined in various articles [10] [12]. B. Orriëns et al. propose a packaging mechanism, named web component, for developing applications by combining various existing Web Services [10]. According to their proposal, the web component would encapsulate the composition logic and the script code in order to combine the services. Their construction would also allow developers to have the various services separated and decoupled, but, in contrast to our proposal, they propound various different stages with their different specification languages. We appoint to a simpler option, in which only one language is needed for the specification of the composition which, moreover, uses a language which is already known and easy to use. Otherwise, G. Piccinelly et al. propose a basic grammar for workflow processes as a language for composing Web Services [12] Although they also provide a graphical environment, it seems to be slightly intuitive as multiple roles have to be assigned to the different services. We believe the developer is more familiarized with his developing language and therefore it may be easier for them to work with an aspect-oriented language, in addition to the aspect-oriented programming advantages.

On the other hand, we discern various articles which propose composition languages based on XML. We can especially outline one proposal which runs in the same line as the different languages proposed on the matter of choreographies and orchestrations [5]. As we mentioned before, there is still not a unified opinion as to which of these languages is more suitable for Web Services composition, nor is there any standard established on this matter. In this sense, we propose an alternative using aspect-oriented techniques instead of XML based languages.

The idea of using aspect-oriented techniques with Web Services is not very widespread, but we can still find some more papers in this sense. Especially, we distinguish a paper focused on modularising Web Service management with AOP [14]. M.A. Verheecke et al. suggest the use of a dynamic aspect-oriented language called JAsCo for decoupling services from the application which invokes them. It is a very interesting proposal in the sense that they redirect abstract requests to Web Services to concrete ones that can be modified in a dynamic way. They focus mainly on the client side, leaning on an intermediate layer called WSML (*Web Services Management Layer)*. In contrast, our proposal uses a general use aspect-oriented language and is not only centred on the client-side, but especially on the server-side, which means that they may not be acting over a Web Service, as the client does not necessarily have to be a service, while we are clearly acting over one. Furthermore, we do not limit to redirecting the services invocations, but to any type of Web Service composition, maintaining the logical order of the operations encapsulated and decoupled from the rest of the application.

5. CONCLUSIONS

The results obtained in this study show how aspect-oriented programming is really useful in order to compose Web Services. In particular, AspectJ has been used for dealing with Web Services orchestrations implementation, and it has proved to improve modularity, scalability and flexibility in these compositions. In addition, aspect-oriented techniques have also been used to implement, extend and reuse the orchestrations interaction patterns.

One of the main advantages of our proposal is the possibility of replacing any of the services in the composition without the need to modify the main application, but only the aspect class; the same can be done for adding new services to the final application.

It is also important to stress the possibility of defining abstract aspects which describe general interaction patterns, which can be later inherited from a concrete aspect which only has to particularize the stubs and the invocation methods to implement the desired composition.

In addition to the orchestration implementation, aspects are also used to add non functional properties to the Web Services [9]. These properties can also be applied to the orchestration itself. In this same line, we are currently working on the choreography implementation, where the interaction pattern and the choreography specification may be aspects themselves.

We would finally like to point out that all the compositions performed with AOP are implemented without the need of knowing the implementation code of the services involved, but only the WSDL document. Besides, all the examples shown in this paper manage compositions from a service, but any of them could be a simple client without our proposal being affected.

6. ACKNOWLEDGMENTS

This work has been developed thanks to the support of CICYT under contract TIC2002-04309-C02-01.

7. REFERENCES

[1] Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leimann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S. *Business Process Execution Language for Web Services Version 1.1*. Microsoft, May 2003.

[2] Arkin, A.. *Business Process Modeling Language (BPML)*. BPMI.org, November, 2002.

[3] Arkin, A., Askary, S., Fordin, S., Jekely, W., Kawaguchi, K., Orchard, D., Pogliani, S., Riemer, K., Struble, S., Takacsi, P , Trickovic, I., Zime, S. *Web Service Choreography Interface (WSCI) 1.0*. Bea System, Sun, Intalio, Sun Microsystems, August 2002.

[4] Cauldwell, P., Chawla, V, Chopra, V., Damschen, G., Dix, C., Hong, T., Norton, F., Ogbuji, U., Olander, G., Richman, M.A., Saunders, K., Zaev, Z.. *XML Web Services*. Wrox Press, 2001

[5] Florescu, D., Grünhagen, A., Kossmann., D. *XL: an XML Programming Language for Web Service Specification and Composition*. Proc 11th International Conference on WWW, Honolulu, Hawai, USA, May 2002

[6] Kavantzas, N. *Web Service Choreography Description Language (WS-CDL) 1.0 Editors draft*. W3C, April 2004

[7] Kiczales, G. *Aspect-Oriented Programming*, ECOOP'97 Conference proceedings, LNCS 1241, June 1997.

[8] Leymann, F. *Web Service Flow Language (WSFL 1.0)*. IBM, May 2001

[9] Ortiz, G., Hernández, J., Clemente, P. J. *Decoupling Non-Functional Properties in Web Services: an Aspect- Oriented Approach*. Workshop EOOWS, ECOOP Conference, Oslo, Norway, June 2004

[10] Orriëns, B., Yang, J., Papazoglou, M.P. *A Framework for Business Rule Driven Web Service Composition*. Workshop ER Conference, Chicago, Illinois, October 2003

[11] Peltz, C. *Web Service Orchestration and Choreography. A look at WSCI and BPEL4WS*. Web Services Journal, July 2003.

[12] Piccinelli, G., Williamns, S.L. *Workflow: A Language for Composing Web Services*. Proc. Int. Conference on Business Process Management, Eindhoven, The Netherlands, June 2003.

[13] Thatte, S.. *XLANG. Web Services for Business Process Design*, Microsoft Corporation, 2001

[14] Verheecke, B., Cibrán, M..A. *AOP for Dynamic Configuration and Management of Web Services*. Proc. International Conference on Web Services, (ICWS-Europe'03) Erfurt, Germany, September 2003.

A Semantic Protocol-Based Approach for Developing Business Processes [*]

Amit Chopra Nirmit Desai Ashok Mallya Leena Wagle Munindar P. Singh
{akchopra, nvdesai, aumallya, lvwagle, singh}@ncsu.edu
Department of Computer Science
North Carolina State University
Raleigh, NC 27695-7535, USA

ABSTRACT

A (business) protocol is a modular, public specification of an interaction among different roles that achieves a desired purpose. We model protocols in terms of the commitments of the participating roles. Commitments enable reasoning about actions, thus allowing the participants to comply with protocols while acting flexibly to exploit opportunities and handle exceptions. A policy is a (typically private) rule-based description of a participant's business logic that controls how it participates in a protocol. We propose that a business process be conceptualized as a cohesive set of protocols, and be enacted by agents playing specified roles in the protocols in which they participate. The agents would respect the given protocols while adhering to their local policies.

We propose OWL-P, a language for specifying protocols, and implement it using a multiagent architecture. We compile OWL-P specifications of protocols into skeletons for each role. Each skeleton corresponds to a set of rules with place-holders for policies. Developing an agent involves using the rules for its intended roles and supplying the necessary policies.

The key benefits of this approach are (1) a separation of concerns between protocols and policies in contrast to traditional monolithic approaches; (2) reusability of protocol specifications based on design abstractions such as specialization and aggregation; and (3) flexibility of enactment of processes in a manner that respects local policies while adapting continually.

This paper develops further results on a programming methodology through which agents can be implemented to realize desired processes. This methodology includes design patterns that ensure that agents built according to those patterns will be guaranteed to be compliant to the stated protocols.

Categories and Subject Descriptors

[**Service Computing & Applications**]: e-Business; [**Software engineering techniques for service-based development**]: Service design principles; [**Core service activities and technologies**]: Service composition; [**Service & AI Computing**]: Multi-agent based service models

[*]This research was sponsored by NSF grant DST-0139037 and a DARPA project.

Keywords

Business Process, Service Composition, Multiagent Systems, Rule-based Systems

1. INTRODUCTION

Business processes typically span multiple *business partners* that are autonomous, and heterogeneous. Business processes involve complex patterns of interactions between the partners. These interactions are organized in the form of business protocols. In the past, business relationships were preconfigured and processes were customized and implemented to suit the partners, as in the Electronic Document Interchange (EDI) approach. However, the EDI approach is not conducive to the development of *open* business processes where partner relationships are developed on the fly. The autonomous and heterogeneous nature of participants poses difficult challenges to the development of such open business processes. This paper presents an approach of developing business processes (for open systems) based on the interaction protocols used and the policies of the partners.

Conceptually, a business process has two important elements, protocols that the partners use to interact, and business policies that drive the partners' enactment of the protocols. Protocols are specifications of interactions and represent the public part of the business process; for the process to be carried out effectively, the partners must adhere to the protocols. By contrast, policies are local to the partners; they capture the internal reasoning of partners. Protocols and policies are related in that a partner's policies drive the execution of the protocols it is participating in. For example, when a protocol allows a participant to choose from multiple actions (messages), the local policy of the participant decides which one to take. Similarly, policies also help decide the contents of the messages sent, and the processing of the messages received. The overall business process is realized as a result of the protocols between the partners.

Flexibility is an important consideration for business processes in open settings. Exceptions and opportunities routinely arise during the course of interactions in such settings. A business process will be flexible if the constituent protocols are flexible. Traditional specifications of protocols such as FSMs and Petri Nets specify a rigid sequences of interactions and lack a high-level semantics. A cornerstone of our approach is the use of commitments to give a declarative semantics to protocols [Yolum and Singh, 2002a]. A commitment is a directed obligation from one partner to another for achieving or maintaining a specified condition. Business protocols can naturally be seen as exchanges of commitments among the parties involved. Therefore, commitments represent an important ingredient of the semantics of business protocols. Flexibility in the protocol comes from reasoning about the commitments and taking actions accordingly.

We define an ontology for protocols using the Web Ontology Language (OWL) [OWL, 2004]. Our ontology is called OWL-P (OWL for Protocols). The ontology provides for concepts such as the roles in a protocol, the messages exchanged between the roles, and declarative rules that describe the effects of sending messages in terms of commitments. OWL-P rules can be converted into Jess rules [Jess] for execution and integrated with policies in a principled way. Our programming model is based primarily on rules. Rules lead to a declarative style of programming where the actual computations are inferred at runtime, thereby enhancing dynamic behavior.

From the software engineering point of view, the clear separation of protocols and policies offers certain advantages. Protocols can be reused across business processes. Protocols may not only be reused directly, they are also amenable to abstractions such as refinement and aggregation [Mallya and Singh, 2004]. However, in our programming methodology, protocol rules consult policies. The integration of policies and protocols at this level encourages a designer to think about the soundness of the business policies with respect to the protocol.

Organization

Section 2 motivates our approach, lists our contributions and the scope of this work, and introduces the basic concepts and terminology. Section 3 describes our proposal for developing protocol specifications and policies. It also describes the system architecture, and the method of generating local flows from an OWL-P specification, with a sketch of correctness proof for the generated local flows. Section 4 compares our work with current research efforts in the area and charts out directions for future work.

2. MOTIVATION

Here, we describe an example of a business process developed with contemporary methodologies and tools, e.g., BPEL [BPEL, 2003] and list their shortcomings. Later sections contrast our approach with current trends to demonstrate the advantages of our approach. Figure 1 depicts a general procurement process where items to be purchased are already selected and the price has been agreed upon. The agents involved in the process are a Customer who wants to buy items, a Merchant who sells items, a Shipper who is a logistics provider, and a Payment Gateway who authorizes payments. The payment-related interactions are imported from the Secure Electronic Transactions (SET) standard [SET, 2003]. Empty circles in the flow of a participant represent the execution of internal business policies, whereas filled circles are the external interfaces through which the participants receive messages. Dark arrows represent the internal control flow. Thus, a sequence of dark arrows, empty circles, and filled circles (in some order) represents the local flow (local process) of the participant. When there are multiple out-edges from empty circles, all of the out-edges are executed in parallel. Since there are multiple participants possibly acting concurrently, the ordering of the messages shown is just one of the possible orders. For example, all the messages after message 8 (messages 9-17) could occur in any order.

Although this process is functionally correct and serves the purpose of its participants, its shortcomings are exposed when examined under the light of service-oriented computing (SOC) environments and open systems. Significant efforts to overcome these have

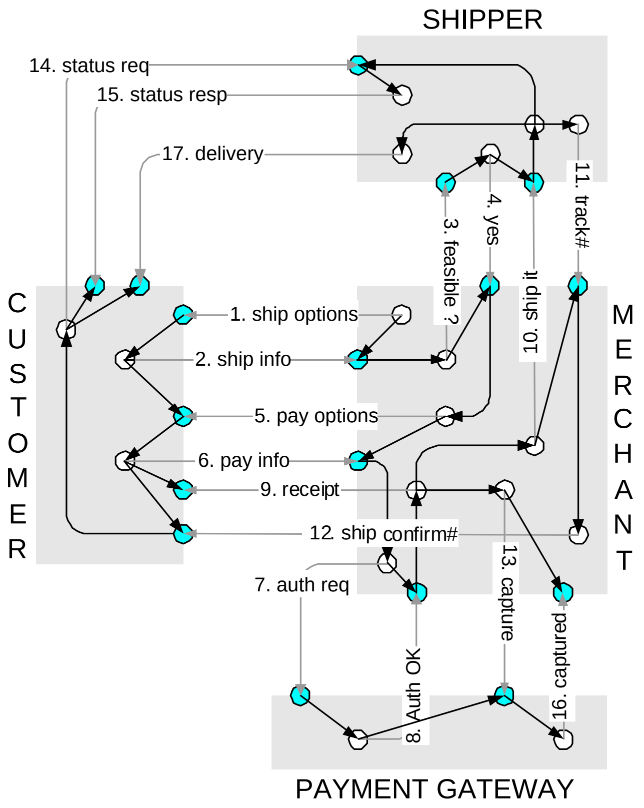


Figure 1: A Business Process

been made by the research community but with limited success. The outline in the earlier section hints at the solution we present. Section 3.6 explains in detail how our approach tackles the challenges of SOC environments.

Lack of Semantics. More often than not, local flows are independently developed by autonomous participants. Although the interfaces of the local flows are exposed in a standard language, e.g., WSDL [WSDL, 2002], no semantics is attached to these interfaces. If the business partners were to be discovered dynamically, as would be the case in open systems, it would be difficult at best to ascertain whether interoperation with them is possible.

Lack of Reusable Components. The local flows of the partners are not reusable. They are monolithic in nature, and formed by ad hoc intertwining of business logic and interactions. Since business logic is proprietary, flows of one partner are not usable by another. If a new customer were to participate in this SOC environment, its local flow would need to be developed from scratch. This is in spite of the fact that interaction patterns are generally reusable.

Aiding Exception Handling. Since in the merchant's local flow, interactions with the payment gateway may be intertwined with interactions with the shipper, a failure on part of one may affect the other. Ideally, as they serve distinct business purposes in the flow, the effect of exceptions of other flows should be minimized.

Runtime Adaptation. Suppose the merchant wishes to change the way it interacts with a customer (maybe because he is a special customer). Say the goods are to be shipped before the payment is received from this customer. Now the local flow cannot adapt to this change at runtime. Also, the customer agent may no longer be able to interact correctly. Similar problems arise in the face of a change of business policy of an agent: it is not clear what updates must be made and where.

2.1 Contributions, Scope, and Significance

Our general contribution is a methodology and design principles for specifying flexible protocols and dynamic processes. Specifically, we show how employing modular protocols can result in reusability and ease of development. We present a novel conceptual model for business processes that allows us to decouple internal local policies of agents from their external interaction behavior. We show how commitments provide the basis for a semantics of the actions of the participants, thereby making the resultant protocols flexible and verifiable. We ground our concepts by introducing an ontology for specifying protocols termed OWL-P. By example, we show how our approach for process enactment is conducive to reuse and aggregation. Finally, we develop a prototype for enacting simple processes having one constituent protocol in a rule-based system.

This paper is a first step in realizing our vision. It emphasizes the modeling and software engineering aspects, and doesn't address technical challenges such as failure and recovery semantics for the protocols, versioning of the protocols and runtime compatibility verification of protocol skeletons. Further, it assumes synchronous communication. Lastly, it defers the composition of multiple protocol skeletons to future work.

The presented work is significant because it yields a new approach for the specification and enactment of business processes. Just as the standardization of network protocols enabled the expansion of the lower layers of Web architectures, business protocols will enable the development of processes for open systems. For this reason, we expect to see an increasing set of business protocols being published, and custom protocols to be designed in narrow domains. This will further increase the significance of our contributions for developing processes based on protocols and local policies.

2.2 Concepts and Terminology

Figure 2 shows our conceptual model for a protocols and policies based treatment of business processes. Boxed rectangles are abstract entities (interfaces), which must be implemented and combined with business policies to yield configurable entities that can be fielded in a running system (rounded rectangles). Abstract entities should be published, shared, and reused among the process developers. We specify a business protocol in terms of a set of rules termed *protocol logic*. Protocol logic encodes the interactions of participating *roles*, who would be bound to specific partners when a process consisting of the given protocol is enacted.

Whereas the protocol logic specifies the protocol from the global perspective, a *protocol skeleton (P-Skel)* specifies the protocol from the perspective of one of the participant roles. Thus, each P-Skel defines the behavior of the respective role with respect to the given protocol.

An *agent* is an implementational entity, representing a real-world, autonomous business partner with its local business rules. An agent may participate in multiple business protocols by adopting a role in each of them. For example, a bookstore may adopt the role of a seller while interacting with customers and the role of a buyer while interacting with publishers. When an agent needs to participate in multiple protocols, a *composite skeleton (C-Skel)* can be created by splicing in the P-Skels, one for each role that the agent plays in the given protocols. For example, in a supply chain process, a supplier would be a merchant when interacting with a retailer in a trading protocol and would be a client in a shipping protocol for sending goods to the retailer. The C-Skel for such a supplier would be composed by splicing in P-Skels for a trading merchant and a shipping customer. This C-Skel specification could be published and then reused for developing other supplier processes.

An agent stipulates its internal business policies in terms of a set of rules we term as *business logic*. The *local flow* of an agent is an executable realization of a C-Skel. Local flow consults the business logic to make decisions. Thus, the combination of a C-Skel with business logic entails the desired local flow, which may be represented in a flow language, e.g., BPEL. A *business process* is the aggregation of the local flows of all the agents participating in it. Conversely, a business process is an implementation of the constituent business protocols.

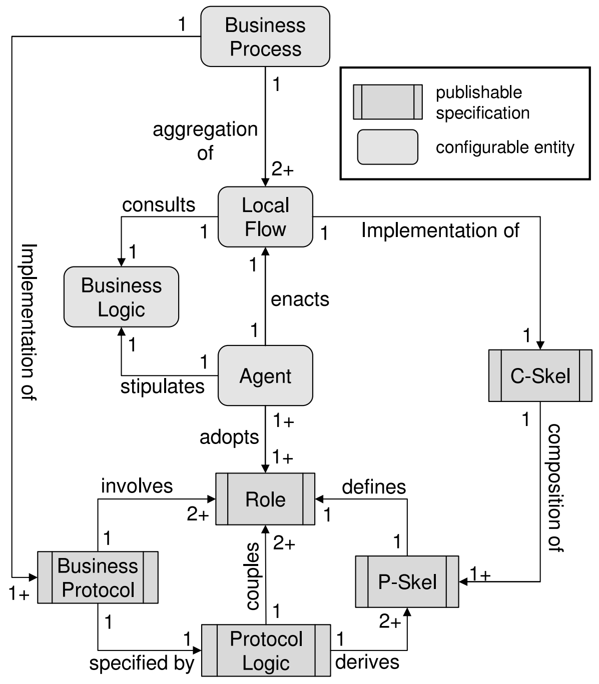


Figure 2: Conceptual Model

2.2.1 Commitments

To talk about how a protocol allows flexibility during enactment presupposes that we can characterize the computations allowed by a protocol and the evolving states of those computations so we can consider whether a particular refinement or detour is legitimate. For business protocols, therefore, this means that we must represent not only the behaviors of the participants but also how the contractual relationships among the participants evolve over the course of an interaction. Doing so enables us to determine if the interactions are indeed compliant with the stated protocols.

The contractual relationships of interest are naturally represented through *commitments*, which have gained importance in the field of multiagent systems [Castelfranchi, 1993]. Commitments capture the obligations of one party to another. For example, the customer's agreement to pay the price for the item after it is delivered is a commitment that the customer has towards the merchant. Using commitments enables us to model not only a participant actions, but also how they advance the ongoing business interaction, which enables us to more readily detect and accommodate business exceptions and opportunities.

Commitments lend coherence to the agents' interactions because they enable agents to plan based on the actions of others. In principle, violations of commitments can be detected and, with the right social relationships, commitments can be enforced—by penalizing participants who do not comply with their commitments. Enforceability of contracts is necessary in practical settings where the participants are autonomous and heterogeneous [Singh, 1998].

To apply commitments presupposes that we are modeling the participants as *agents*. Agents naturally represent parties such as the business partners involved in an e-business scenario, who might collaborate but retain their autonomy. Commitments have been used to model interaction protocols, especially in e-business settings [Verdicchio and Colombetti, 2002, Yolum and Singh, 2002b]. Such formulations generally afford more flexibility to the participants in choosing the actions they may take at different stages. Much of our technical development is based on established concepts of distributed computing and temporal logic. Since commitments might be unfamiliar to some readers, we introduce them first.

DEFINITION 1. *A commitment* $\mathsf{C}(x, y, p)$ *denotes that the agent* x *is responsible to the agent* y *for bringing about the condition* p*.*

Here x is called the *debtor*, y the *creditor*, and p the *condition* of the commitment. The condition is expressed in a suitable formal language.

Commitments can also be *conditional*, denoted by $\mathsf{CC}(x, y, p, q)$, meaning that x is committed to y to bring about q if p holds. For example, the conditional commitment $\mathsf{CC}(c, b, \mathit{goods}_g, \mathit{pay}_p)$ means that the customer c is committed to pay the bookstore b an amount p if the bookstore delivers the book g to the customer. When the bookstore delivers the goods, i.e., when the goods_g proposition holds, the conditional commitment $\mathsf{CC}(c, b, \mathit{goods}_g, \mathit{pay}_p)$ is automatically converted into the base-level commitment $\mathsf{C}(c, b, \mathit{pay}_p)$.

2.2.2 Commitment Operations

Commitments are created, satisfied, and transformed in certain ways. The following operations are conventionally defined for commitments [Singh, 1999].

1. CREATE(X, C) establishes the commitment *c* in the system. This can only be performed by *c*'s debtor *x*.

2. CANCEL(X, C) cancels the commitment *c*. This can only be performed by *c*'s debtor *x*. Generally, cancellation is compensated by making another commitment.

3. RELEASE(Y, C) releases *c*'s debtor *x* from commitment c. This only can be performed by the creditor *y*.

4. ASSIGN(Y, Z, C) replaces *y* with *z* as *c*'s creditor.

5. DELEGATE(X, Z, C) replaces *x* with *z* as the *c*'s debtor.

6. DISCHARGE(X,C) c's debtor x fulfills the commitment.

A commitment is said to be *active* if it has been created, but not yet discharged.

3. APPROACH

Protocols need to have a clear definition and their specification should use terminology that is understood by all the participants. The intent of drawing up an unambiguous protocol specification is to reap the benefits of standardization as has been done in other areas of computer science. Unlike networking protocols, business protocols need to be specified at a higher level of abstraction so that the protocol designers are not bogged down by low-level details of the protocol execution and can focus on the business aspects. To further ease the job of business protocol designers, our framework separates the local business logic and other legal concerns from the public protocol. Such a separation of concerns makes for better engineered processes that enable reuse and are easy to understand. This section describes our approach for specifying flexible, commitment-based protocols and how we separate local policies from protocols.

3.1 Protocols

A protocol has a unique identifier. The protocol ontology defines the following elements, most of which are shown in Figure 3.

1. *Roles* that participate in the protocol. Each protocol has at least two roles.

2. *Messages* that the roles use to communicate with each other, along with message formats and meanings. The message formats detail various attributes that a message can have. For example, the `requestForQuote` message in the Netbill protocol, which is sent by the customer to the merchant asking for a price quote on a certain item, has one attribute for identifying the item, one attribute to identify the sender, and one for the receiver.

3. *Preconditions* and *effects* for each message, so that the participants derive the same meaning from a message and, consequently, have their locally maintained protocol states in sync. As a result, preconditions and effects provide unambiguous meanings to messages. Preconditions on messages enforce a (partial) ordering of the messages, allowing participants to detect a violation of the protocol. A message can be sent only if its precondition holds. If a message is received in a state at which the preconditions for that message do not hold, the sender of that message is in violation of the protocol. Effects of messages are actions that the participants should take to keep their states consistent. Message effects are realized through *actions* as described below. Message preconditions are evaluated in conjunction with the participant's policy. Therefore, an important aspect of the participants' freedom of enactment is that the policy can block a message precondition by returning a `false` value and potentially violating a protocol. This topic is discussed in Section 3.2.

4. *Commitments* and domain specific *propositions* that are needed by the participants to keep track of events in the protocol. Commitments are needed to identify and enforce obligations between participants.

5. *Actions* that the participants should perform as part of the effects of a message. Note that some of these actions are *internal* to the participant. That is, the participant takes these actions locally. Although messages have effects, participants are allowed to check with their policies before taking these actions. The actions are as follows.

   - Adding or removing a proposition from the state to keep track of the protocol execution. Messages between participants change the protocol state by adding or removing propositions from the working memory.
   - Invoking a procedure. Effect of a message might be invoking a local procedure.

- Creating a commitment. A message may require creating commitments. For example, in the Netbill protocol, when the merchant sends a quote to the customer, the merchant is making a conditional commitment to the customer that it will send the goods if the customer promise to pay for them. This is a local action.
- Invoking another protocol. A participant may choose to invoke another protocol, which nests inside the current protocol. This is not a local action, since a new protocol would involve other participants. A detailed treatment of protocol nesting and a theoretical framework for the same is provided in [Mallya and Singh, 2004].
- Sending or receiving a message

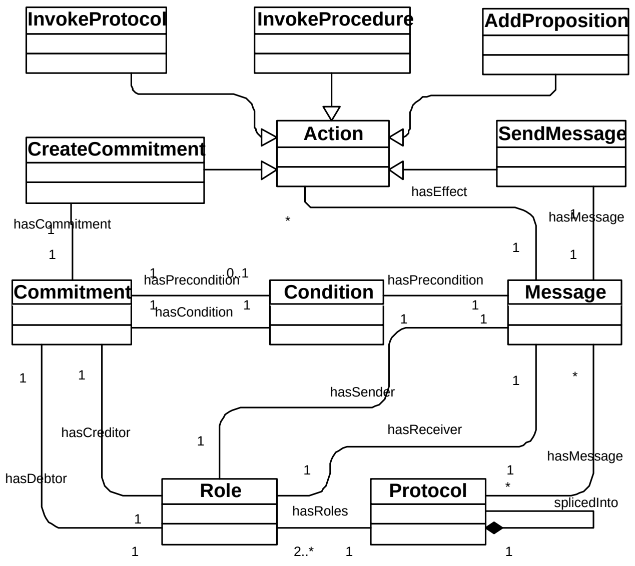


Figure 3: UML Model for OWL-P

The OWL-P ontology, in its RDF/XML serialization, is available at http://bombadil.csc.ncsu.edu/owl/Protocol.owl. The NetBill protocol, which was developed by Sirbu [Sirbu, 1997] as a method of selling goods online, has been described using this ontology, and is available at http://bombadil.csc.ncsu.edu/owl/Netbill.owl. Figures 6 and 8 show precondition-effect rules of the escrow protocol.

3.2 Policies

Since protocols specify only the key interactions, the participants of a protocol now have the liberty to enact the protocol details as they deem fit. Such freedom of execution is required by business protocols because business interactions are governed to a large extent by the (internal) business policies of the partners, and other external factors such as trade laws, rules, and regulations. For example, an online trading protocol that only requires the exchange of goods for money allows its participants to perform the goods transfer and the money transfer in any mutual order. It is up to the local policy of the merchant if it decides to send the goods before receiving money.

Policies are consulted before firing any precondition-effect rule, by evaluating the precondition in conjunction with the policy. Because of this, poorly specified policies could cause a violation of a protocol. For this reason, it is important that the designers of policies follow certain guidelines so that the policies agree with protocols. We identify two kinds of effects: *required* and *optional*. These serve as a guide to design policies that do not force protocol violations.

- *Required effects*. All effects that involve a commitment operation on a base-level (i.e., unconditional) commitment are required effects. If policies respect required effects, there will never be a deliberate violation of commitment. Even if participants disagree on what domain propositions hold, they will never disagree on what commitments exist, and what commitments have been discharged. Since base-level commitments are the foundations for detecting protocol violations, policies should not disable message effects that operate on base-level commitments.

- *Optional effects*. All other effects are optional and may be disabled by local policies in the preconditions. For example, the creation of a conditional commitment is an optional effect, since no obligation is violated if one participant creates a conditional commitment and another participant does not recognize the creation of the conditional commitment. Note that if a conditional commitment gets converted into a base level commitment, then that conversion is a required effect, because a base level commitment is being created.

3.3 System Architecture

Figure 4 shows our process enactment framework. Its components fall into two broad categories: those that are in the local domain of the participant and those that are in the public domain. In Figure 4, the horizontal dashed line demarcates the local and public domains. The protocol specification lies in the public domain. The messages that are sent between participants pass through the public domain, although these will be secure or encrypted messages in most real-life situations. The local domain contains all components of the participant agent's architecture. Th thin arrows connect components that the participants implement with inputs from the OWL-P specification. The thick arrow denotes the messaging service used by participants to communicate with others.

Public Components. The OWL-P specification of a protocol is available to all potential participants via a URI. The specification was explained in Section 3.1 and is not repeated here.

Local Components. The local components of each participant include a knowledge base, a rule base, and a program that sends and receives messages. The knowledge base is a store for commitments and propositions. The rule base is generated from message preconditions and effects by the algorithm given in Figure 7. The rule base acts on the data contained within the knowledge base. The rule base is consulted by the main messaging interface when messages are received and in turn can send messages proactively (based on the policies by invoking the messaging interface.)

Our current implementation uses the Java Expert System Shell (JESS) to implement the rule base. JESS maintains the knowledge base as part of its execution.

3.4 Enactment

Here, we present an algorithm for generating role skeletons from an OWL-P protocol specification. To better demonstrate this algorithm, we present an OWL-P specification for the Escrow protocol [Escrow.com, 2003], which is a three-party protocol, and the generated skeleton corresponding to the Buyer role of the Escrow protocol. Figure 5 shows the Escrow protocol in a state-based representation where the Buyer has already selected the items and the

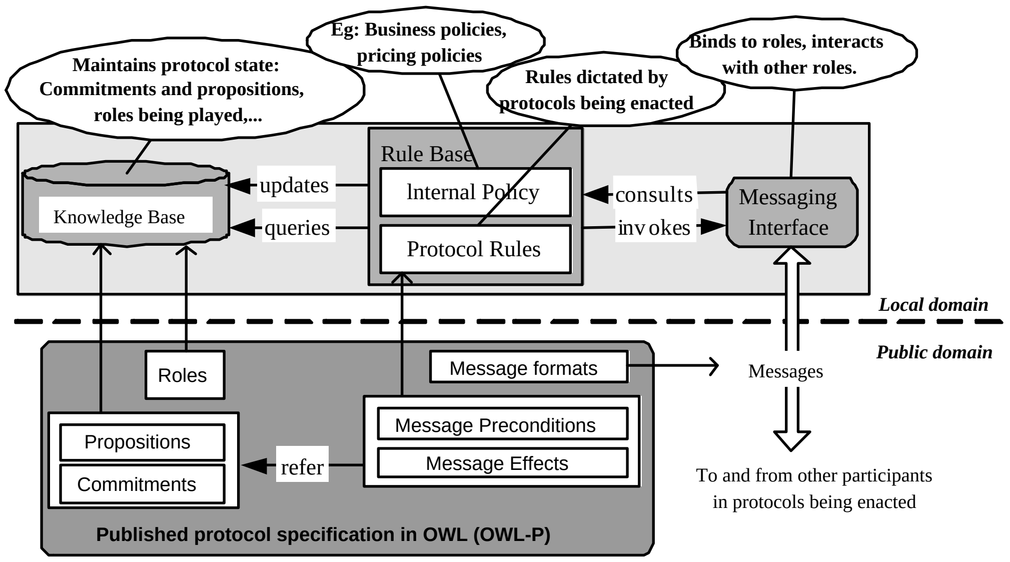

Maintains protocol state: Commitments and propositions, roles being played,...
Eg: Business policies, pricing policies
Rules dictated by protocols being enacted
Binds to roles, interacts with other roles.
Knowledge Base
updates
queries
Rule Base
Internal Policy
Protocol Rules
consults
invokes
Messaging Interface
Local domain
Public domain
Roles
Message formats
Messages
Propositions
Commitments
refer
Message Preconditions
Message Effects
To and from other participants in protocols being enacted
Published protocol specification in OWL (OWL-P)

Figure 4: Policy-Based Protocol Enactment Framework

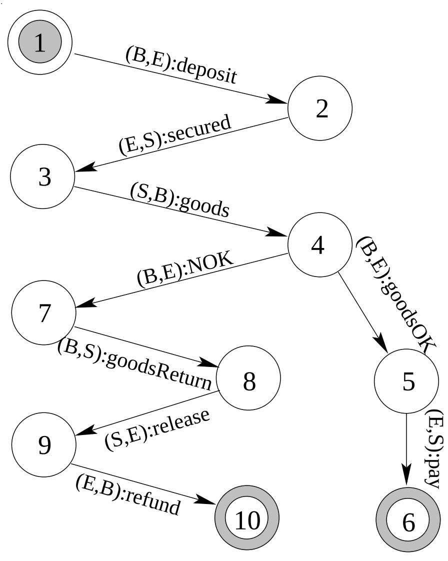


Figure 5: The Escrow Protocol

```
(B, E): deposit
    start_p ⇒ sendMessage(deposit)
        addProposition("deposit_p")
        createCommitment(S, B, secured_p, goods_p)
        createCommitment(E, B, released_p, refund_p)
(E, S): secured
    deposit_p ⇒ sendMessage(secured)
        addProposition("secured_p")
        createCommitment(E, S, goodsOK_p, payment_p)
(S, B): goods
    secured_p ⇒ sendMessage(goods)
        addProposition("goods_p")
        createCommitment(B, S, goodsNOK_p, goodsReturn_p)
(B, E): goodsOK
    goods_p AND policyOK_p ⇒ sendMessage(goodsOK)
        addProposition("goodsOK_p")
(B, E): goodsNOK
    goods_p AND ¬policyOK_p ⇒ sendMessage(goodsNOK)
        addProposition("goodsNOK_p")
        createCommitment(S, B, goodsReturn_p, release_p)
(B, S): goodsReturn
    goodsNOK_p ⇒ sendMessage(goodsReturn)
        addProposition("goodsReturn_p")
(S, E): release
    goodsReturn_p ⇒ sendMessage(release)
        addProposition("release_p")
(E, B): refund
    release_p ⇒ sendMessage(refund)
        addProposition("refund_p")
(E, S): payment
    goodsOK_p ⇒ sendMessage(payment)
        addProposition("payment_p")
```

Figure 6: OWL-P for the Escrow Protocol

Escrow is aware of the amount of payment. Figure 6 is the corresponding OWL-P specification in convenient notations after abstracting away the details about message formats.

Let us formally define a Protocol in OWL-P. A Protocol $\mathbb{P}$ defines a set of messages $\mathbb{M}$ and a set of participant roles $\mathbb{R}$. Hence, $\mathbb{P}$ = ($\mathbb{M}$, $\mathbb{R}$). A message $M \in \mathbb{M}$ is defined as M = (ρ_s, ρ_r, C_{pre}, Eff) where, $\rho_s \in \mathbb{R}$ is the sender, $\rho_r \in \mathbb{R}$ is the receiver, C_{pre} is the precondition of the message and Eff is the set of all actions being effects of the message. A skeleton of a role ρ then, is $\mathbb{P}_\rho$ where $\mathbb{P}_\rho$ = ($\mathbb{M}_\rho$, $\mathbb{R}_\rho$). $\mathbb{M}_\rho$ is the set of all messages in which ρ is involved and $\mathbb{R}_\rho$ is the set of all the roles with whom ρ interacts including ρ itself. Figure 7 describes the algorithm. Desai and Singh developed a similar algorithm in [2004 (To Appear] which was based on state-based representations to specify protocols. However, it had no semantics attached with the actions and states. Note that OWL-P describes the protocol from a global perspective where the propositions are added to a global state and there are no distributed sites. The role skeletons describe the protocol from the perspective of the corresponding participant. When the Partition algorithm is run through the OWL-P for the

```
Partition($\rho$)
  $\mathbb{M}_\rho \leftarrow \phi$
  For all $M \in \mathbb{M}$
    If $\rho = \rho_s$ Then
      $\mathbb{M}_\rho \leftarrow \mathbb{M}_\rho \cup M$
    Else If $\rho = \rho_r$ Then
      ModifyPrecondition($C_{pre}$)
      If sendMessage $\in$ $Eff$ Then
        Replace sendMessage with ReceiveMessage
      $\mathbb{M}_\rho \leftarrow \mathbb{M}_\rho \cup M$

ModifyPrecondition(cond)
  If cond is an atomic proposition Then
    FindReplacement(cond)
    return
  Else
    For all components cond_comp of cond
      ModifyPrecondition(cond_comp)

FindReplacement(cond)
  For all $M(\rho_s, \rho_r, C_{pre}, Eff) \in \mathbb{M}$ such that
    addProposition(cond) $\in$ $Eff$
    If $\rho = \rho_s \vee \rho = \rho_r$ Then
      return
    Else
      cond $\leftarrow C_{pre}$
      ModifyPrecondition(cond)
```

Figure 7: Partition Algorithm: Generating a Role Skeleton from OWL-P

Buyer role in the Escrow protocol, it generates the skeleton for the Buyer role as shown in Figure 8. Notice that for the goods message the precondition `secured_p` is replaced by `deposit_p` and for the refund message the precondition `release_p` is replaced by `goodsReturn_p`. This is due to the fact that when the Buyer receives these messages, it will not know about `secured_p` and `release_p`. This is because the Buyer is not involved in the interactions that cause those propositions to hold, namely, `secured` and `release`. Replaced propositions are found by the Partition algorithm. Procedures `ModifyPrecondition` and `FindReplacement` take a reference to a condition expression and replace its constituent propositions. Also, the `sendMessage` actions for the messages that the Buyer receives are replaced by `receiveMessage` actions.

Prototype Implementation. We showed how to generate skeletons for roles involved in one protocol. But more often than not, agents play roles in more than one protocol in a business process. To enact the local flow of such an agent, its role skeleton for each protocol it participates in should be generated from the protocol's OWL-P specification. Next, the skeletons need to be spliced in together to generate a composite skeleton. Finally, to enact the local flow, the composite skeleton needs to be augmented with the agent's local policies. This paper does not address the methodology for splicing in multiple role skeletons. Figure 9 shows the constituent protocols and the participants involved in a procurement business process from the global view. The entities shown are the participants and the edges are the protocols through which they interact. As a simple guideline, each participant will need as many role skeletons as the number of edges attached to it.

We enact the role skeletons generated by the Partition algorithm

```
(B, E): deposit
    start_p ⇒ sendMessage(deposit)
        addProposition("deposit_p")
        createCommitment(S, B, secured_p, goods_p)
        createCommitment(E, B, released_p, refund_p)
(S, B): goods
    deposit_p ⇒ receiveMessage(goods)
        addProposition("goods_p")
        createCommitment(B, S, goodsNOK_p, goodsReturn_p)
(B, E): goodsOK
    goods_p AND policyOK_p ⇒ sendMessage(goodsOK)
        addProposition("goodsOK_p")
(B, E): goodsNOK
    goods_p AND ¬policyOK_p ⇒ sendMessage(goodsNOK)
        addProposition("goodsNOK_p")
        createCommitment(S, B, goodsReturn_p, release_p)
(B, S): goodsReturn
    goodsNOK_p ⇒ sendMessage(goodsReturn)
        addProposition("goodsReturn_p")
(E, B): refund
    goodsReturn_p ⇒ receiveMessage(refund)
        addProposition("refund_p")
```

Figure 8: Skeleton for the Buyer role in Escrow Protocol

in terms of Jess Rules. Jess rules can be directly mapped from the message rules in the protocol skeletons. The messaging system is implemented in Java. Jess rules invoke Java methods to send messages. On the receiving side the messaging system receives messages and dispatches events to the Jess rule-base. Our messaging system is implemented in JADE (Java Agent Development Framework).

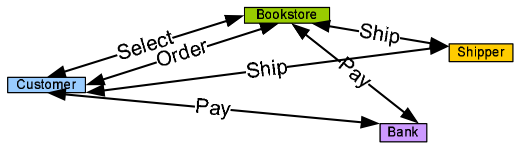


Figure 9: A Procuement Business Process Global View

3.5 Sketch of Correctness

Here, we informally argue that the skeletons generated by our algorithm are sound and complete with respect to the OWL-P specification. That is, the generated skeletons when executed together, realize the same computations as the input OWL-P and realize no other computations. We define a computation C as a sequence of message exchanges observed in a run of the protocol. So, $C = M_0 \cdot M_1 \cdot M_2 \cdots M_n$. Note that we do not consider the local policies as part of an OWL-P computation as they are not specified in OWL-P.

3.5.1 *Soundness*

We show that if a computation C is allowed by the generated skeletons then OWL-P also allows it. Let C be allowed by the skeletons. Let $M_i \cdot M_j$ be a step in C. Let ρ_1 be the sender of M_i and ρ_2 be the receiver. From the view of ρ_2, the next observable event will be sendMessage action in effects of M_i. Let that message be M_j and hence ρ_2 be the sender of M_j.

Now, because $M_i \cdot M_j$ is allowed by the skeletons, the following properties hold:

1. In the local state of ρ_1, precondition for sending M_i holds.
2. In the local state of ρ_2, precondition for receiving M_i holds.
3. After receiving M_i, the local state of ρ_2 enables the precondition of sending M_j.

If we can show that these conditions also hold for OWL-P, it means that $M_i \cdot M_j$ is allowed by OWL-P.

By line 4 of the Partition algorithm, in $\mathbb{P}_{\rho_s}$, the precondition of M_i is added unchanged from $\mathbb{M}$ because ρ_s is the sender of M_i. So Condition 1 above holds for OWL-P.

All the propositions that hold in the local state of ρ_2 also hold for OWL-P as ρ_2 was involved in the latest interaction and OWL-P is the global view of the protocol and there are no distributed states. Hence, if the precondition for receiving M_i holds for ρ_2, it must hold for OWL-P. So, Condition 2 above holds for OWL-P.

Notice that while generating a skeleton, for the receiving role of a message, no effects are modified except for replacing sendMessage actions by receiveMessage actions. Hence, the local state of ρ_2 after receiving the message is identical to the OWL-P view of the state. Because the precondition for sending M_j is enabled for ρ_2, it will also be enabled for OWL-P; hence Condition 3 above holds.

3.5.2 Completeness

We show that if a computation C is allowed by OWL-P then the generated skeletons also allow it. Let C be allowed by OWL-P. Let $M_i \cdot M_j$ be a step in C. Let ρ_1 be the sender of M_i and ρ_2 be the receiver. Obviously, ρ_2 is the sender of M_j.

Here, the three conditions listed in Section 3.5.1, already hold for OWL-P. If we can show that they also hold for the generated skeletons then $M_i \cdot M_j$ will also be allowed by the skeletons. Condition 1 holds as discussed for Section 3.5.1.

Due to the procedure FindReplacement in the Partition algorithm, C_{pre} of M_i in $\mathbb{P}_{\rho_2}$ is modified such that the propositions unknown to ρ_2 are replaced by the propositions as last known to ρ_2 that lead to M_i. Hence, if the C_{pre} holds for OWL-P, it will also hold for $\mathbb{P}_{\rho_2}$. Hence Condition 2 holds. Condition 3 holds as discussed for Section 3.5.1.

3.6 Revisiting the Motivating Problems

In this section we show how and to what extent does our approach aid in solving the problems identified in Section 2.

Semantics for Protocols. Our protocol specification is minimal in that it does not dictate all possible computations or runs. The protocol specifies the meanings of messages in terms of when they can be sent and what their effects are. Message sequencing is achieved as a consequence of planning on the part of the agents to reach a final state of the protocol. Participants can therefore use their policies to enact local flows that best serve their interests. Hence, when partnerships are formed dynamically, the role skeletons of the partners can be verified against each other. Because the protocols are standardized and published, skeletons generated from them are guaranteed to be compatible. However, if the participants have different versions of the protocol, or they customized the generated skeletons, their semantics must be reasoned about to verify the compatibility. We defer the discussion of compatibility verification to future work.

Reusable Protocols. The clear separation between protocols and policies allows for the specification of protocols independent of local policies. Hence, the OWL-P protocol specifications are modular and fit well into software engineering abstractions such as refinement and aggregation. As these specifications are published, designers can reuse them to build local flows of new agents by generating corresponding role skeletons and augmenting them with their local policies.

Runtime Adaptability. The use of commitments to represent important events in the protocol allows a protocol to generate a variety of runs. Protocols can be refined into other protocols that constrain the runs more, or can be adapted into other protocols during runtime [Mallya and Singh, 2004]. Hence, if a change of business policy or interaction pattern occurs, modifying the rule base at runtime and reverifying the compatibility with the partners would allow the local flows to adapt at runtime.

Exception Handling. Commitments allow us to identify which agent is responsible for which event. Commitments bring accountability and enable agents to intelligently reason about exceptions. For example, an agent can decide to release another from a commitment if the need arises or if the debtor agent asks for an extension of the deadline of the commitment. Pending base-level commitments correspond to an absolute obligation of one agent to the other. Any failure at a point where base-level commitments are pending may result in undesirable results. Hence, protecting the scopes for the life-time of base-level commitments is a guideline to the designer.

4. DISCUSSION

Developing business processes for open systems poses significant challenges. Interactions in business processes can be quite complex making interoperation of autonomous partners a major concern in open systems. Most research efforts focus on alleviating the interoperation problems by documenting an expected choreography of message exchanges. The Web Services Choreography Interface (WSCI) [WSCI, 2002] specification describe the choreography of message exchanges, using control flow constructs. As such, they can support complex interactions. However, a WSCI specification gives no semantics to interactions, which makes it less amenable to reuse. Business Process Execution Language (BPEL) [BPEL, 2003] is a flow language designed to specify composition of Web services. However, a BPEL specification intertwines business logic and interactions, and therefore cannot be reused and lack the semantics of the activities. Also, BPEL specifications are not published; it represents the internal orchestration of a partner's local flow.

The Semantic Web Services Initiative has produced OWL-S [DAML-S, 2002]. OWL-S is in essence similar to WSCI, however service interfaces are given semantics in the model. In addition, OWL-S processes can be dynamically composed via planning. But the specifications are logically centralized and support the perspective of only one participant thereby limiting autonomy of others.

The RosettaNet [RosettaNet, 1998] effort centers around publishing protocols and designing the business processes around them. RosettaNet represents a step in the right direction. However, they are currently limited to two-party request-response interactions called Partner Interface Processes (PIPs). PIPs lack a semantics.

Whereas the local flow of a partner cannot be reused by another, the protocols can be. At the same time, given a formal semantics, protocols can refined or aggregated, thus yielding new protocols. Mallya and Singh [Mallya and Singh, 2004] treat these concepts formally. The Business Process Handbook [Malone et al., 2003], in a similar vein, catalogues different kinds of business processes in a hierarchy. For example, *sell* is a generic business process. It can be qualified by *sell what*, *sell to who*, and so on. Our notion of a protocol hierarchy bears similarity with the Handbook. Our effort is focused on providing formal semantics to the hierarchy.

Older technologies such as workflow systems lacked the flexibility and agility that current business processes need. Businesses today are moving their processes online, with the help of the Semantic web [Petrie and Sahai, 2004]. It is this area of confluence of business processes and the Semantic Web that we deal with. Our work draws from traditional concepts in distributed computing and multiagent systems, and some of our results apply to agent interaction protocols.

The representation of business contracts is an interesting area of research. Grosof and Poon [2003] represent agent contracts in OWL and RuleML. They develop an ontology for processes and contracts. Davulcu *et al.* [2004] develop a logic for specifying contracts in Web services. Commitment-based protocols serve precisely the purpose of specifying contractual arrangements. Commitments are, in principle, enforceable and partners can be held responsible for violation of commitments. It is reasonable to expect that not all parts of a complicated contract will be reflected in the protocol, because contracts are sensitive to the context in which the interactions take place. For example, the Uniform Commercial Code (UCC) [UCC] has certain stipulations for merchants and consumers. Such codes are more like policies than protocols. Some partners may be required to follow the UCC, perhaps based on their geographical location. How to layer and prioritize such policies is a direction of future research.

Future Directions. One important direction is the layering of contextual policies such as UCC, as described above. Contextual policies play an important role in a partner's decision-making and would presumably be given a higher priority. That is to say a local policy should behave differently under different contexts.

Another vital direction to explore is the OWL-P representation of composite protocols. To enact the local flow of a merchant in Figure 1, we need to splice in the skeletons for payment protocol, the shipping protocol and the procurement protocol in such a way that they realize the local flow as seen in Figure 1. Also, there might be dependencies between the protocol skeletons that are spliced in. How should we specify the data flow dependencies, control dependencies, failure-recovery dependencies and the relationship of one protocol to the other in OWL-P? Also, when a pair of agents dynamically form partnerships but have different versions of protocols, how can we verify their compatibility?

References

BPEL. Business process execution language for web services, version 1.1, May 2003. www-106.ibm.com/developerworks/webservices/library/ws-bpel.

Cristiano Castelfranchi. Commitments: From individual intentions to groups and organizations. In *Proceedings of the AAAI-93 Workshop on AI and Theories of Groups and Organizations: Conceptual and Empirical Research*, 1993.

DAML-S. DAML-S: Web service description for the semantic Web. In *Proceedings of the 1st International Semantic Web Conference (ISWC)*, July 2002. Authored by the DAML Services Coalition, which consists of (alphabetically) Anupriya Ankolekar, Mark Burstein, Jerry R. Hobbs, Ora Lassila, David L. Martin, Drew McDermott, Sheila A. McIlraith, Srini Narayanan, Massimo Paolucci, Terry R. Payne and Katia Sycara.

H. Davulcu, M. Kifer, and I.V. Ramakrishnan. Ctr-s: A logic for specifying contracts in semantic web services. In *Proceedings of the 13th International World Wide Web Conference*, May 2004.

Nirmit Desai and Munindar P. Singh. Protocol-based business process modeling and enactment. In *International Conference on Web Services*, 2004 (To Appear). http://www4.ncsu.edu/∼nvdesai/icws04.pdf.

Escrow.com. Online escrow process, 2003. http://www.escrow.com/solutions/escrow/process.asp.

Benjamin N. Grosof and Terrence C. Poon. SweetDeal: Representing agent contracts with exceptions using XML rules, ontologies, and process descriptions. In *Proceedings 12th International Conference on the World Wide Web*, 2003.

Jess. http://herzberg.ca.sandia.gov/jess/.

Ashok U. Mallya and Munindar P. Singh. A semantic approach for designing commitment protocols. In *Proceedings of the AAMAS-04 Workshop on Agent Communication*, July 2004. To appear.

Thomas W. Malone, Kevin Crowston, and George A. Herman, editors. *Organizing Business Knowledge: The MIT Process Handbook*. MIT Press, Cambridge, MA, 2003.

OWL. Web ontology language, Feb 2004. http://www.w3.org/TR/2004/REC-owl-features-20040210/.

Charles Petrie and Akhil Sahai. Business processes on the web. *IEEE Internet Computing*, 8(1):28–29, January 2004.

RosettaNet. Home page, 1998. www.rosettanet.org.

SET. Secure electronic transactions (SET) specifications, 2003. http://www.setco.org/ set_ specifications.html.

Munindar P. Singh. Agent communication languages: Rethinking the principles. *IEEE Computer*, 31(12):40–47, December 1998.

Munindar P. Singh. An ontology for commitments in multiagent systems: Toward a unification of normative concepts. *Artificial Intelligence and Law*, 7:97–113, 1999.

Marvin A. Sirbu. Credits and debits on the Internet. *IEEE Spectrum*, 34(2):23–29, February 1997.

UCC. http://www.law.cornell.edu/ucc/ucc.table.html.

Mario Verdicchio and Marco Colombetti. Commitments for agent-based supply chain management. *ACM SIGecom Exchanges*, 3 (1):13–23, 2002.

WSCI. Web service choreography interface 1.0, July 2002. wwws.sun.com/ software/ xml/ developers/ wsci/ wsci-spec-10.pdf.

WSDL. Web Services Description Language, 2002. http://www.w3.org/TR/wsdl.

Pınar Yolum and Munindar P. Singh. Commitment machines. In *Proceedings of the 8th International Workshop on Agent Theories, Architectures, and Languages (ATAL-01)*, pages 235–247. Springer-Verlag, 2002a.

Pınar Yolum and Munindar P. Singh. Flexible protocol specification and execution: Applying event calculus planning using commitments. In *Proceedings of the 1st International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 527–534. ACM Press, July 2002b.

A Service Oriented Architecture for Advertising Games

P. Avesani, M. Cova, R. Tiella
ITC-irst
via Sommarive 18
38050 Povo, Italy
{avesani,cova,tiella}@itc.it

A. Sharma
Birla Institute of Technology
Department of Computer Science
835215 Ranchi, India
arun_bit123@rediffmail.com

ABSTRACT

A critical issue of distributed systems is concerned with the advertising task. Current solutions require an ex-ante agreement on a common shared language. Although such an approach is feasible from the technological point of view, it is not effective in practice. The process of managing this agreement may present social implications that make the solution difficult to achieve. Recent trends in research propose a new approach based on advertising games where the agreement on a common language is produced at run time. Nevertheless up to now such a model has been studied only through simulations with standalone platforms. Our contribution is the design and the development of the first web services oriented architecture for advertising games. Therefore we approached all the issues typical of distributed systems neglected by the simulators like asynchronous communications, denial of services, and so on. Finally we present a real world application where the architecture has been deployed to support the advertising task using an advertising game model.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Distributed Applications; H.3.5 [**Online Information Services**]: Web-based Services

General Terms

Design

Keywords

Web Services, Advertising, Language Games, Interoperability, BPEL4WS

1. INTRODUCTION

The development of distributed systems is a complex task that becomes even more complex if one adopts the open world assumption where single peers can autonomously design and deploy their own services. In such a scenario interoperability arises as a critical issue because of the heterogeneity of services specifications.

Advertising is one of the steps that is affected by heterogeneity. Usually advertising is concerned with two tasks: service discovering and service semantics specification. In the following we will focus on service semantics specifications.

When new services are published there is the need to notify the specification of service contents. The lack of a shared representation language makes this step harder. The semantic web scientific community is currently working on this issue. Mainly its research effort is pursuing two alternative strategies: promoting a reference ontology [5, 9] or supporting the reconciliation of different ontologies [3, 7, 11]. The first approach, simple from the technological point of view, is not feasible in practice due to sociological implications. The second approach requires a significant manual effort because a full automated reconciliation of ontologies is too complex.

More recently a new research trend proposes an alternative approach to overcome the need of ex-ante agreement in favor of an ex-post agreement. The basic idea is to support a run time process of negotiation that enables the convergence to a common lexicon [15]. Differently from semantic web, the advertising game model aims to shift the problem from matching of representations to negotiation of lexicon. Lexicon can be conceived as an association list that maps labels to meanings, i.e. the service content specifications. Of course a shared lexicon is not equivalent to an expressive knowledge representation language. In fact, a lexicon supports only a common set of symbols to refer to a collection of objects or categories.

There are other research initiatives to support the automated negotiation of mapping between two information sources but these approaches don't scale because they build pairwise mapping [4, 12].

Naming games [14, 16], and more specifically advertising game [1], have been proposed to support the negotiation of a common lexicon independently from the number of players. Up to now the studies on advertising games have been proved to be effective in dealing with the critical issues of service advertising [2]. Simulations have been developed to test hypothetical scenarios of distributed recommendation systems.

The main drawback of these preliminary results is that they are achieved through simulations on standalone platform. No one has approached the issue of porting the model based on advertising game into a full service oriented architecture.

The main contribution of this work is the design and the implementation of a fully distributed, service oriented architecture to enable a real world deployment of an advertising game model. With this approach we dealt with many issues neglected by standalone platforms, like asynchronous communications, denial of services and many others.

The proposed architecture has been deployed to support a sharing annotation system among distributed blog servers. In this specific application the advertising game is used to align the references of distributed ski route catalogs. A first trial is currently under test in the domain of ski mountaineering. Nevertheless, it is not a goal of this paper a detailed presentation of such an application.

In the next section we briefly summarize the basic concepts of advertising games. After that we illustrate the service oriented architecture that has been designed to support the deployment of the advertising game model in practice. Finally we show a real world application where such an architecture has been deployed to support the advertising task in a fully distributed systems.

2. GAMING APPROACH TO ADVERTISING PROBLEM

An advertising game involves two or more peers. The basic interaction involves two peers with different roles: consumer (or speaker) and provider (or hearer), therefore a session of communication is not symmetric. Nevertheless each peer can play different roles in different sessions.

More formally an advertising game is defined by a set of peers $\mathcal{P}$, of size $\mathcal{N}_{\mathcal{P}}$ where each peer $p \in \mathcal{P}$ has a set of concepts $\mathcal{C}_p = \{c_1, \ldots, c_n\}$ of size $\mathcal{N}_{\mathcal{C}}$. A set of objects is defined, $\mathcal{O} = \{o_1, \ldots, o_m\}$, such that a subset of them can be conceived as representative of a given concept. The objects are shared among the peers while concepts are private of each peer. A lexicon $\mathcal{L}$ is a relation between concepts and words, where it is assumed that words are composed using a shared and finite alphabet. Lexicon is extended with a couple of additional information: the number of times the relation has been used and the number of times the relation was in successful use. Each peer $p \in \mathcal{P}$ has its own lexicon drawn from the Cartesian product $\mathcal{L}_p = \mathcal{C}_p \times \mathcal{W} \times \mathcal{N} \times \mathcal{N}$, where $\mathcal{W}$ is a set of words and $\mathcal{N}$ the natural numbers to represent the peers' preferences. The lexicon may include synonymous words, two words associated to the same concepts, and homonymous words, the same word can be associated to two different concepts. A peer $p \in \mathcal{P}$ is then defined as a pair $p =< \mathcal{L}_p, \mathcal{C}_p >$.

An advertising game is an iterative process where at each step two peers are selected to interact together. The interaction proceeds as follows (see Figure 1). First the speaker p_s randomly selects a concept from its set of concepts, then it encodes the concept $c_s \in \mathcal{C}_s$ through a word w_j. The word is chosen accordingly to the current version of the local lexicon $\mathcal{L}_s$ (local to speaker p_s). The denotation of concept c_s is obtained looking at the most successful word. A word w_j is more successful than a word w_k iff $< c_s, w_j, u_j, a_j >\in \mathcal{L}_s$, $< c_s, w_k, u_k, a_k >\in \mathcal{L}_s$, $u_j \geq u_k$ and either $a_j/u_j > a_k/u_k$ or $a_j/u_j = a_k/u_k$ and $u_j > u_k$, where u_j represents how many times the word w_j has been used and a_j represents how many times there was an agreement on word w_j with other peers. In case of a tie, a random choice is performed. The hearer p_h decodes the word w_j retrieving the associated concept $c_h \in \mathcal{C}_h$ looking at its own lexicon $\mathcal{L}_h$.

The next step is concerned with the actuation task. Actuation can be modeled as a function $f_a : \mathcal{C} \longrightarrow 2^{\mathcal{O}}$ that takes in input a concept and gives in output a subsample of objects. Actuation function has a stochastic component, therefore two subsequent invocations of $f_a(c_h)$ do not necessarily produce the same outcome. The outcome of actuation is sent back to the speaker. Once received a sample of objects, the speaker has to deal with the perception task. Perception can be modeled as a function $f_p : 2^{\mathcal{O}} \longrightarrow \mathcal{C}$ that takes in input a sample of objects and gives in output an hypothesis of concept, namely $\hat{c}_h$, that may subsume such a sample. Of course the hypothesis formulated by the perception function is sensitive to the size of the sample.

The last step is concerned with the assessment. The speaker has to verify whether the concept perceived from the hearer's objects is the same selected at the beginning of the communication session. The assessment process can now be carried on easily checking the condition $c_s = \hat{c}_h$.

If the concept referred by the hearer is the same selected by the speaker, both of them give a positive reinforcement to their lexica updating the corresponding word-concept association as follows: $< c_s, w_j, u_j + 1, a_j + 1 >\in \mathcal{L}_s$ and $< c_h, w_j, u_j + 1, a_j + 1 >\in \mathcal{L}_h$. If the hearer replies with a different concept $c_s \neq \hat{c}_h$, it means that the communication failed, the peers' lexicon is updated with a negative reinforcement increasing only the counters of lexical relation (while the counters of agreements on the lexical relation remain the same): $< c_s, w_j, u_j + 1, a_j >\in \mathcal{L}_s$ and $< c_h, w_j, u_j + 1, a_j >\in \mathcal{L}_h$.

3. AN ARCHITECTURE FOR ADVERTISING GAME

Now that we have laid out the theoretical basics of the advertising game technique, it is time to discuss the architecture we propose for its concrete realization. While some implementations already exist, e.g. [13], they only consist of stand-alone simulators that lack the distributed nature of the model. The architecture presented here, by contrast, brings the model into a fully distributed environment, where its strength and weak points can thoroughly and more realistically be evaluated.

In fact, the main goal that guided the design of our architecture was to support an open-ended distributed system. The architecture promotes heterogeneity (by only specifying the minimal set of requirements needed for interoperability and allowing for alternative solutions to be adopted in many parts of the system), autonomy (by avoiding strong or centralized coordination among parts of the system), and robustness to evolution (by taking into account that the system is inherently dynamic and subject to change).

By providing an implementation of the advertising game technique, we target legacy distributed systems, where peers are interconnected and cooperating, but have no or little ability to interoperate in terms of lexicon alignment, as explained in the introduction. Therefore, from the software engineering point of view, the main challenge of our research is to realize a component that can be transparently plugged into an existing distributed system, requiring no or as few as possible modifications to its legacy parts. We decided

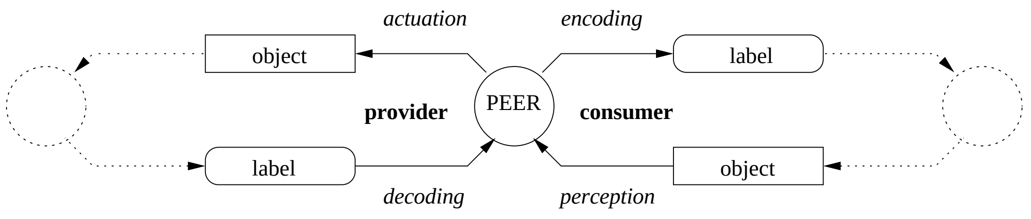


Figure 1: The schema shows the four basic actions performed by a peer during the advertising game, according to its role in the game (provider or consumer).

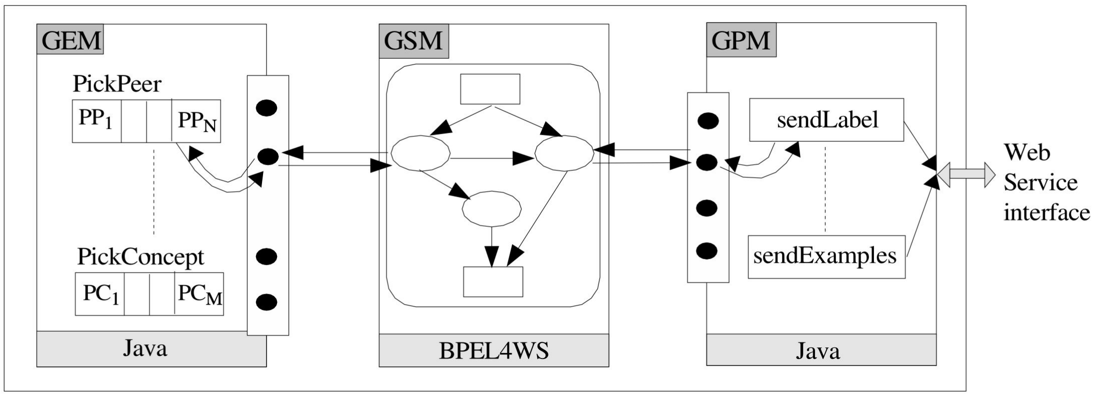


Figure 2: The architecture of the component that implements the advertising game technique.

to focus especially on web services based systems, because they represent the current solution of choice for distributed systems interoperability.

The component that implements and encapsulates the advertising game technique is made up of three modules: the *Game Engine Module* (GEM), the *Game Strategy Module* (GSM), and the *Game Protocol Module* (GPM). The GEM module provides the set of primitives needed to perform the advertising game, the GSM module models and realizes the strategy used during a game, the GPM module provides communication primitives. GEM and GPM are providers of mechanisms: they implement and export the building blocks of the advertising game technique. The GSM module, by contrast, realizes policies: it specifies which of these building blocks should be used and how they should be combined to perform a game. The key advantage obtained by this division of the system is that it allows to clearly separate communication protocols from gaming strategies. As a consequence, only protocols need to be defined at system level, while strategies can be autonomously decided by each peer. Furthermore, changes in one module do not affect other modules as long as the common interface remains constant. A graphical description of the architecture is given in Figure 2.

We start describing the component architecture from the Game Strategy Module. The GSM module represents the core part of this architecture. It encompasses the "intelligence" of the peer and informs its acts during the game. It is in charge of a number of activities: deciding when to start a new game and how long it is to last, what are the concepts to play with, which peers are to be taken as opponents, how to evaluate other peers' performance, what type of feedback to provide on current lexicon on the ground of past game results.

At this point of our research, different choices for each of these activities seem to be reasonable and worth more experimentation. It is therefore critical being able to easily implement, test, and compare different alternatives. While one can think of many technologies to model a strategy and implement its runtime, our favor went to BPEL4WS. BPEL4WS is a composition language normally used to perform web services orchestration. It fits our requirements with its built-in coordination features, relatively high-level flow control constructs, and the ability to use services offered by external components through standard invocation interfaces. Other appealing features of BPEL4WS are its relatively high level of abstraction, the possibility to do quick, graphical programming, and its flexibility.

BPEL4WS processes realize strategies selecting and orchestrating primitive services offered by the GEM and the GPM modules [1]. In fact, a typical step in a strategy requires the following operations: choose from the primitives offered by GEM one that provides the desired functionality, e.g., choose the peer to be challenged next according to a particular selection method, for instance, randomly. Then, invoke the primitive and collect the result, e.g., get information about the peer, in particular its address. Use this information as intended, e.g., leverage the information about

[1]Some BPEL4WS engines adopt the WSIF [8] framework which allows to define Java bindings to invoke services. When available, we leveraged this technology to minimize inter-module communication cost.

```
<sequence>
  <invoke name="choosePeer" partner="GEM"
    portType="GemPT" operation="pickPeerAtRandom"
    outputContainer="PeerInfo" />
  <assign><copy>
    <from variable="PeerInfo" part="PeerURL" />
    <to variable="Message" part="PeerURL" />
  </copy></assign>
  <invoke name="chooseConcept" partner="GEM"
    portType="GemPT" outputContainer="ConceptInfo"
    operation="pickConceptAtRandom" />
  <invoke name="denote" partner="GEM"
    portType="GemPT" operation="denote"
    outputContainer="LabelInfo" />
  <assign><copy>
    <from variable="LabelInfo" part="Label" />
    <to variable="Message" part="Label" />
  </copy></assign>
  <invoke name="sendLabel" partner="GPM"
    portType="GpmPT" operation="sendLabel"
    inputContainer="Message"
    outputContainer="Examples" />
</sequence>
```

Figure 3: BPEL4WS code that implements a step of the strategy.

the chosen peer to guide the selection of the concept to play with. Lastly, communicate, if needed, to a remote peer via one of the primitives defined in GPM.

This step of the strategy may be described by the following pseudo-code snippet:

```
Peer p := GEM.pickPeerAtRandom();
Concept c := GEM.pickConceptAtRandom();
Label l := GEM.denote(c);
sendLabel(p, l);
```

Its BPEL4WS implementation is shown in Figure 3.

The GPM module embeds the *choreography* [2] of the distributed system: it defines the protocols available for inter-peer collaboration in terms of sequences of communication primitives.

We identified two fundamental protocol variants: pull-based and push-based protocols. In the pull-based protocol, the speaker chooses the label to play with, sends it to the hearer and waits for a set of examples. Finally, the perception step allows the speaker to align its lexicon to the hearer's lexicon. By contrast, in the push-based protocol, the speaker chooses the label to play with and a set of examples and sends them to the hearer. The hearer then performs the perception step and updates its lexicon to align it with the speaker's. Symmetrical feedback protocols are also possible: in this case, the variants discussed above are extended with a final feedback message, from the speaker to the hearer in the pull-based version, from the hearer to the speaker in the push-based version.

Accordingly with this sketch of the protocols, the GPM module provides primitives

- To send a peer a label;
- To send a label and a set of examples;
- To send a set of examples;
- To send a feedback message.

Higher level characteristics of the communication protocol, such as timeout settings and exception conditions handling, are left to be decided to the strategy module. For full details on the communication protocol, refer to [6].

Support and enforcement of choreography represent the minimal requirement to be satisfied for interoperability. In other words, as long as an implementation of the GPM primitives is provided, alternative realizations of the advertising game technique can be developed, even adopting different design or technologies, and still maintain interoperability with the architecture we propose.

The last module of the advertising game component is the Game Engine Module. The game engine has to provide an implementation of the operations that are used internally by a peer during a game. In particular, it offers methods

- To select concepts to play with;
- To select peers to be challenged;
- To perform the reification step;
- To perform the denotation step;
- To perform the perception step;
- To update the lexicon on the basis of game results;
- To extend the lexicon if a new label or concept are learnt during the game.

These basic activities can be performed in many different ways. As an example, let consider the activity of selecting the peer to play with the next game. One can imagine alternative criteria to guide this process: e.g., random selection among the set of known peers, selection on the basis of the number of past contacts or past performances, or more sophisticated selection techniques such as active sampling. The GEM module collects and makes available the implementations of each different criterion to the strategy module.

Consequently, it is clear that the GEM module has no requirement of minimality, as it was for the communication primitives. Instead, it provides an open-ended collection of activity realizations that can grow with the necessity of devising more complex strategies. At first glance, for each basic activity there seems to be a limitless number of alternative realizations. However, we expect that further research on advertising game strategy will help single out those criteria that really are helpful for building a shared lexicon and, thus, that the number of implementations needed will be quite limited.

To get a feeling of the primitives that the GEM module might make available, Table 1 shows some variants for the activities of perception, concept and peer selection.

[2]We use the term "choreography" as defined in the Web Services Choreography Description Language draft [10]:

> [Choreography] defines from a global viewpoint [web services'] common and complementary observable behavior, where message exchanges occur, when the jointly agreed ordering rules are satisfied.

Activity Family	*Variants*	*Input*	*Output*
perceive	perceivePairwiseMatching	Concept, ExampleSet	int
	perceiveMachineLearning	Concept, ExampleSet	int
pickConcept	pickConceptAtRandom	Peer	Concept
	pickConceptAfterPeer		Concept
	pickMostUsedConcept		Concept
	pickLeastUsedConcept		Concept
pickPeer	pickPeerAtRandom	Concept	Peer
	pickPeerActiveSampling		Peer
	pickPeerAfterConcept		Peer
	pickMostContactedPeer		Peer
	pickMostFriendlyPeer		Peer

Table 1: An excerpt of the primitives made available by the GEM module.

It should be further noted that different peers can have completely disjointed sets of gaming primitives, e.g., one adopts the random peer selection mechanism, the other opts for a selection based on the history of past games. The only requirement is that every peer has at least one realization for each activity, i.e., each peer must be provided with an operation to choose the next game opponent.

While minimality and standardization of communication primitives guarantee interoperability of peers, extendability and specialization of game engines promote differences in peers' personalities and lexicon building behaviors.

Differences in the purpose of modules is reflected in the technology used to implement them. To encode the business logic, we use Java, because it offers a convenient environment to implement computationally demanding activities required by some game primitives. We have already discussed the advantages of using BPEL4WS to define the games strategies. Lastly, for the communication protocol, we turned to web services technology to benefit from its interoperability, reusability and deployability characteristics.

4. A REAL WORLD APPLICATION

The technique and the architecture described in the previous sections are currently under test in a real-world application in the domain of ski mountaineering.

It is common for ski mountaineers to form on-line communities. A community is aggregated around a web site that generally offers two kinds of services: a ski route catalog and a ski trip annotation list. Ski routes are mainly concerned with persistent, static and validated information while ski trips are usually volatile, fresh and not certified.

The ski routes catalog provides information about ski routes. For each ski route it generally gives geographical information, e.g., starting and ending point of the route, a measure of the its difficulty, and other details that might be useful, e.g., an estimate of the time needed to complete it. Members of the community are encouraged to write comments about their excursions on routes contained in the site catalog. These annotations take the form of a report of a trip and provide information about dynamic aspects of the route, e.g., snow conditions, presence of dangers, etc. Annotations are collected in the ski trip annotation list and made available to all members of the community.

The typical use case for the services offered by a similar site is as follows. A ski mountaineer browses the ski route catalog to identify a number of candidate routes for her next trip. Her choice among all possible destinations is guided by her reading of other users' annotations. For example, she might decide not take a trip if another ski mountaineer signaled in his trip annotation the danger of avalanches on that route.

In the scenario described so far, ski mountaineers have access only to the annotation list provided by their own community. However, many ski mountaineering communities exist and their catalogs present large overlapping sections, i.e., the same routes are recorded on multiple sites. Therefore, annotations about trips done along the same routes are produced in different communities. Nevertheless, it is important to maximize the sharing of past trip experiences in order to achieve better trip planning and safer ski trips. Thus, there is a need for inter-community annotations exchanging and sharing.

Aggregation services are designed to solve this problem. Essentially, an aggregation service collects annotations from different sources and builds a reverse mapping between annotations and annotated items. That is, given a certain item, it allows to access all known annotations referring to it.

It is clear that to make this approach effective the aggregation service must have a way to understand that different annotations, generally originating from different sources, refer to the same item. In other words, items must be globally identified. As an example, let consider the case of an aggregator serving a certain number of readers communities, where annotations consist of comments about books. In this domain, the ISBN number represents a global identifier of a book. Therefore, as long as every annotation uses the ISBN number to identify the book it refers to, the aggregator can easily associate a book to its comments.

Unfortunately, a unique global identifier is not available for most domains. In particular, not only there is no global reference catalog for ski routes, but also an effort in this direction is not foreseeable in the future. What is needed then, in order to make possible the sharing and aggregation of ski trip annotations, is a method to dynamically build a common ski route catalog that does not require a standardization or agreement effort.

The model based on the advertising game technique and the service oriented architecture that we illustrated in previous sections satisfy this need. In particular, the advertising game technique can be leveraged to originate a common reference system for ski routes without requiring any kind of ex-ante agreement among different ski mountaineering systems.

This common reference system allows single communities to preserve their autonomy in the design of ski route schemas, while, at the same time, permits ski mountaineers to effectively access trip annotations independently from the specific catalog they refer to.

Let examine how the advertising model maps to the ski mountaineering domain. Ski mountaineering web sites play the role of peers. Ski routes map to concepts. They are private to each peer, in the sense that a peer is free to model a route adopting the schema it prefers. As a consequence, generally, different sites represent the same routes in different ways. The role of objects is played by concrete representations of ski route models. A common choice to represent a ski route is to provide an XML linearization of the information available for the route. For example, Figure 4 shows such linearization for the same route as modeled by two different ski mountaineering sites.

```
<item>
  <id>5947</id>
  <top>Monte Cevedale (Zufallspitze)</top>
  <region>Ortles</region>
  <title>Dalla Vedretta di Solda</title>
  <global_difficulty>PD+</global_difficulty>
  <ski_difficulty>S3</ski_difficulty>
  <base_height>2600</base_height>
  <top_height>3757</top_height>
  <gap>1300</gap>
  <exposure>NW</exposure>
</item>

<route>
  <trip_id>2109</trip_id>
  <end_p>Cevedale (Monte)</end_p>
  <start_p>da Solda</start_p>
  <area>Alto Adige</area>
  <district>null</district>
  <valley>Valle di Solda - Suden tal</valley>
  <difficolty>BSA</difficolty>
  <exposure>N</exposure>
  <start_h>2610</start_h>
  <end_h>3769</end_h>
  <gap>1159</gap>
  <start_place>Solda, Funivia di Solda</start_place>
</route>
```

Figure 4: Representation of the same route on two different ski mountaineering web sites.

A critical step in the advertising game is represented by the assessment task. It is in charge of evaluating whether two route linearizations represent the same ski route, possibly modeled using different schemas. At the moment the assessment task is performed using a bipartite matching algorithm. The linearizations are divided in tokens and schema information is dropped. This leaves with two sets of tokens, in our example {*5947, Monte Cevedale (Zufallspitze), Ortles, Dalla Vedretta di Solda, PD+, S3, 2600, 3757, 1300, NW*} and {*2109, Cevedale (Monte), da Solda, Alto Adige, null, Valle di Solda - Suden tal, BSA, N, 2610, 3769, 1159, Solda, Funivia di Solda*}. A bipartite matching algorithm is then used, given a distance function, to find the optimal matching of tokens. The assessment ends successfully if the evaluation of the matching is above a given threshold. More experimentation is underway to improve and tune the algorithm.

It should be clear, then, that the advertising game is completely independent from ski route schemas and dependent only on route information. The underlying assumption is that while the modeling of a ski route can be done in many different ways, i.e., there is high variance on schema models, the information that describes a route is rather homogeneous.

5. CONCLUSIONS AND FUTURE WORK

The paper focuses the attention on a recent approach to advertising based on the notion of advertising games. After a brief summary of the basic concepts of advertising games, we have introduced a service oriented architecture to support a real distributed implementation of such a model. We argued how the design choices are compliant with the requirements of distributed systems. More in detail, we aimed to minimize the global assumption enabling autonomous local design choices.

The next step will be concerned with the evaluation of the architecture in a real world setting that has been recently deployed in the domain of ski mountaineering. We are currently testing it in a scenario that involves three web sites: www.moleskiing.it, www.skirando.ch and www.gulliver.it.

From the technological point of view we are particularly interested in assessing whether BPEL4WS is effective in developing flexible alternative strategies for evolutionary advertising games.

6. ACKNOWLEDGEMENTS

We would like to thank Paolo Busetta for his suggestion on BPEL4WS as a powerful tool for game strategy encoding. Alessandro Agostini and Paolo Busetta helped us to revise this paper.

This work was partially funded by the DiCA project, thanks to a grant of INRM (Istituto Nazionale Ricerca Montagna) and PAT (Provincia Autonoma di Trento).

7. REFERENCES

[1] A. Agostini and P. Avesani. Advertising games for web services. In R. Meersman, Z. Tari, and D. Schmit, editors, *Eleventh International Conference on Cooperative Information Systems (CoopIS-03)*, Berlin Heidelberg, 2003. Springer-Verlag LNCS.

[2] P. Avesani and A. Agostini. A peer-to-peer advertising game. In M. Orlowksa, M. Papazoglou, S. Weerawarana, and J. Yang, editors, *First International Conference on Service Oriented Computing (ICSOC-03)*, pages 28–42, Berlin Heidelberg, 2003. Springer-Verlag LNCS 2910.

[3] S. Bergamaschi, S. Castano, and M. Vincini. Semantic integration of semistructured and structured data sources. *SIGMOD Record*, 28(1):54–59, 1999.

[4] P. Bouquet, L. Serafini, and S. Zanobini. Semantic coordination: a new approach and an application. In *Second International Semantic Web Conference*, volume 2870 of *Lecture Notes in Computer Science*, pages 130–145. Springer Verlag, September 2003.

[5] J. Broekstra, M. C. A. Klein, S. Decker, D. Fensel, F. van Harmelen, and I. Horrocks. Enabling knowledge

representation on the web by extending RDF schema. In *World Wide Web*, pages 467–478, 2001.

[6] M. Cova, A. Sharma, and R. Tiella. Specification of a service oriented architecture for advertising games. Technical Report T04-06-01, ITC-irst, June 2004.

[7] A. Doan, P. Domingos, and A. Y. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *SIGMOD Conference*, 2001.

[8] M. J. Duftler, N. K. Mukhi, A. Slominski, and S. W. ana. Web Services Invocation Framework (WSIF). In *OOPSLA 2001 Workshop on Object-Oriented Web Services*, 2001.

[9] J. Hendler and D. McGuinness. Darpa agent markup language. *IEEE Intelligent Systems*, 15(6), 2000.

[10] N. Kavantzas, D. Burdett, and G. Ritzinger. Web services choreography description language version 1.0. `http://www.w3.org/TR/2004/WD-ws-cdl-10-20040427/`, 27 April 2004. W3C Working Draft.

[11] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic schema matching with cupid. In *The VLDB Journal*, pages 49–58, 2001.

[12] B. Magnini, L. Serafini, and M. Speranza. Linguistic based matching of local ontologies. In *Workshop on Meaning Negotiation (MeaN-02)*, Edmonton, Alberta, Canada, July 2002.

[13] A. McIntyre. Babel: A testbed for research in origins of language. In *Proceedings of COLING-ACL 98*, Montreal, 1998. ACL.

[14] L. Steels. Grounding symbols through evolutionary language games. In A. Cangelosi and D. Parisi, editors, *Simulating the evolution of language*, pages 211–226. Springer Verlag, London, 2001.

[15] L. Steels and F. Kaplan. Bootstrapping grounded word semantics. In T. Briscoe, editor, *Linguistic evolution through language acquisition: formal and computational models*, chapter 3, pages 53–73. Cambridge University Press, Cambridge, 2002.

[16] L. Steels and A. McIntyre. Spatially distributed naming games. *Advances in complex systems*, 1(4), January 1999.

Flexible Matchmaking of Web Services Using DAML-S Ontologies

Antonio Brogi
Dept. of Computer Science
University of Pisa, Italy

Sara Corfini
Dept. of Computer Science
University of Pisa, Italy

Razvan Popescu [*]
Dept. of Computer Science
University of Pisa, Italy

ABSTRACT

Service discovery is one of the major challenges in the emerging area of Service Oriented Computing (SOC), which promotes the notion of service as the basic brick for the development of next generation distributed heterogeneous software systems. State-of-the-art matchmaking algorithms for Web services range from algorithms based on the DAML-S Service Profile to improved ones based on the DAML-S Service Model. In this paper we propose a new technique for Web service discovery which features a flexible matchmaking by exploiting DAML-S ontologies. Our algorithm allows for partially matched services to be discovered by addressing issues such as: (1) fine-grained matchmaking at the level of atomic processes rather than at the entire service level as in previous approaches, (2) multiple runs of services, and (3) the fact that non-trivial requests can only be satisfied collectively by a set of services rather than by a single execution of a single service. In this way we extend the matchmaking process between queries and advertisements from one-to-one to one-to-many.

Categories and Subject Descriptors

H.3.5 [**Information Storage and Retrieval**]: Online Information Services—*Web-based services*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods

General Terms

Algorithms

Keywords

Web service discovery, matchmaking algorithms, DAML-S ontologies

[*]Corresponding author: via F. Buonarroti 2, 56100 Pisa, Italy. Email: popescu@di.unipi.it

1. INTRODUCTION

SOC [12] is a computing paradigm which uses services as building blocks for developing applications. Basically, services should be technology neutral, loosely coupled components and should support location transparency so as to be optimally exposed by providers and easily discovered by requesters (clients). Such services vary from simple ones such as returning the postal code given a town, to complex ones such as selling airline tickets/shopping portals. SOC relies on the Service Oriented Architecture (SOA) for creating such an infrastructure in which services are published, discovered and executed with minimal programming effort. The basic SOA implies providers which expose services to a service registry and requesters which discover these services and execute them, both through the use of service descriptions. A service description minimally contains information about the functionalities provided by the service but it can further be augmented with behavioural information as in e.g. [1, 4, 6, 10] as well as other type of information such as Quality of Service (QoS), preconditions, effects and so on.

The World Wide Web Consortium (W3C) as well as other standards bodies strongly support a Web service model having three component roles – users, providers and registries – in which providers advertise their services to the registries, from where users further discover such services. The Universal Description Discovery and Integration (UDDI) is currently the only universally accepted standard for Web service discovery and it consists of a specification for defining a service registry of available Web services similarly to a global electronic yellow pages. As noted by [5], "*currently Web services are simple and static*" and standards for Web services that can be dynamically discovered and composed are still to follow. Moreover, complex Web services based interactions between businesses require more than SOAP, WSDL and UDDI can offer, hence future Web services standards will have to cope through others with Web Services Description Languages (that should extend WSDL with behavioural elements such as QoS, preconditions and effects), Choreography, Orchestration as well as workflow, negotiation and management standards.

(Web) service discovery is the first major problem to be tackled and it is often referred to as (Web) service matchmaking. Matchmaking is the process that takes as input a query specifying the inputs and outputs (IOs) of a desired service as well as a service registry consisting of (service) advertisements and that gives as output a list of matched services. UDDI uses a keyword based matching system that often gives poor performance. Search engines are a good example for showing how inaccurate keyword based search for documents can be. Matchmaking is a fundamental problem that SOC has to deal with and a vital component of the SOA. It also proves to be very helpful in the process of (semi-)automatic composition of services. During the last years, increasing attention has been devoted to the problem of Web service matchmaking. One of the state-of-the-art matchmaking algorithms has been described by Bansal and Vidal in [3]. Their algorithm takes as input a query with the IOs of the desired Web service as well as a Web service repository and it searches for a service able to satisfy the query as a result of a single run. For each query, it specifies whether the IOs match as well as the overall (at the service level) matching degree. The matching degree can be `exact`, `plug-in`, `subsumes` or `failed`, depending on the ontological relations between matched IOs.

In this paper we propose an extension of the algorithm introduced by Bansal and Vidal [2, 3] by relaxing the matchmaking process into allowing queries to be satisfied not only by one service but collectively by a set of services. Our motivation comes from the fact that it is likely to have queries which can only be satisfied by composing several advertised services. We may think for example of a client desiring to plan his travel for the holidays by booking flight tickets as well as hotel accommodation while taking into account various parameters such as weather, season prices and so on. In this paper we deal with the process of discovering such services.

Our main contribution can be summarized into the following four points:

1. Designing a *fine-grained matchmaking* at the level of atomic processes rather than at the level of the entire process as in [2, 3].
2. Considering *multiple runs of services.* [2, 3] take into account one execution only of a service advertisement while verifying the fulfillment of the request. We argue that multiple execution of advertisements is a must for satisfying particular requests. For example, [2, 3] match as `failed` an advertisement which produces *either* o_1 *or* o_2 as output given a request for a service producing *both* o_1 *and* o_2. Still, by executing the advertisement twice, one may get *both* o_1 *and* o_2 in a transparent manner with respect to the requester.
3. Extending the matchmaking process by returning *partial matches.* As mentioned before, the matching algorithm described by Bansal and Vidal searches one service capable of satisfying the request by itself. As one can imagine, this is not applicable for non trivial services (applications) and requests so there is need for discovering (sub)services which partially satisfy the request. This proves to be useful in order to individuate the set of all possible (sub)processes which may later be used in order to generate a service composition able to satisfy the request.
4. Extending the matchmaking process between queries and service advertisements from one-to-one to *one-to-many.* We argue that a query should not be coupled with one service as in [2, 3] but with a set of services which complementarily are able to fulfill it.

The rest of the paper is organised as follows. In Section 2 we present the current state-of-the art in Web services matchmaking followed in Section 3 by the description of our flexible matchmaking algorithm. Finally, in Section 4 we draw some conclusions and outline future directions of investigation.

2. WEB SERVICES MATCHMAKING

As mentioned in the Introduction, using UDDI for service discovery often leads to inaccurate matches, hence researchers have been trying to develop new matchmaking technologies. Most notable approaches use Semantic Web based techniques for the process of matchmaking. The current state-of-the-art in Web services matchmaking has been set by researchers dealing with matchmaking based on DAML-S [8] ontologies. DAML-S is an ontology for describing Web services and it is written in DAML-OIL. The root of the ontology is represented by the generic class Service which has three subclasses: Service Profile (which says "what the service does"), Service Model ("how the service works") and Service Grounding (which describes "how to access the service").

The Service Model has a Process Model subclass which provides a view of a Web service in terms of process compositions. DAML-S defines three types of processes: atomic, simple and composite. An atomic process is a process that is executed in a single step (from the point of view of the user client of the service). It can not be decomposed further and it has an associated grounding. A simple process is similar to an atomic one but it can not be invoked directly and it does not have an associated grounding. It is used to provide an abstract view of a set of processes (hence executed in a single step) that is, a simplified view of a composite process. A simple process is equivalent to a Black Box. Finally, a composite process consists of other processes, the composition being made with the following control constructs: `split`, `sequence`, `unordered`, `split+join`, `choice`, `if-then-else`, `iterate` and `repeat-until/while`. A composite process may also be thought at as a Glass Box.

It is important to note that atomic processes only have defined IOs. The IOs set corresponding to a composite process is given by the IOs sets of its subprocesses by taking into account the process type. For example a `choice` process with two atomic subprocesses having each one output – o_1 and o_2 respectively – is able to generate as output *either* o_1 *or* o_2 but *not both* o_1 *and* o_2. Dually, a `choice` process with two atomic subprocesses having each one input – i_1 and i_2 respectively – is able to execute if *either* i_1 *or* i_2 or *both* i_1 *and* i_2 are provided. For example, the matchmaking based on the Service Profile only (similar somehow to matching two Black Boxes) allows for the match of a service request which asks for two outputs o_1 *and* o_2 with a service advertisement which provides *either* o_1 *or* o_2 but *not necessarily both* o_1 *and* o_2 (e.g., a `choice` process), while the actual request can be left unsatisfied.

Due to the fact that the Service Profile sees Web services through their IOs only, matchmaking strategies based on the Service Profile have been proven to have some limitations (as mentioned above) and, as a natural extension, algorithms for matchmaking based on the Service Model have been developed in order to better match the functionalities of Web services.

Although most DAML-S based approaches deal with matchmaking Web services at the Service Profile level (e.g., [11]), Bansal and Vidal presented in [2, 3] an improvement to the matchmaking process by using an algorithm based on the DAML-S Process Model. Their algorithm takes as input a query specifying the desired IOs as well as a repository of DAML-S Web services and returns the degree of match (i.e., `exact`, `plug-in`, `subsumes` or `failed`) by taking into account the Process Model trees of the advertisements as well as the ontological relations between matched IOs. According to [2, 3], a service request R matches a service advertisement A if every input $A(input_i)$ of the advertisement has a corresponding matching input $R(input_j)$ in the request and similarly, every output $R(output_k)$ of the request has a corresponding matching output $A(output_l)$ in the advertisement. In other words, the request should provide all the inputs (possibly more) for the advertisement while the advertisement should give all the outputs (possibly more) to the request. Based on the semantic equivalence between two concepts being matched, one may have:

- an `exact` match (e.g., the requester wants a "DVD" and the provider gives it a "Digital Versatile Disk")
- a `plug-in` match (e.g., the requester wants "British Music DVDs" and the provider gives it "Music DVDs")
- a `subsumes` match (e.g., the requester wants "Music DVDs" and the provider gives it "British Music DVDs"), or
- a `failed` match (e.g., the requester wants a "DVD" and the provider gives it a "MC")

The algorithm of Bansal and Vidal stores DAML-S service advertisements as trees, each tree corresponding to a service Process Model. Each node in the tree (other than leaf nodes) corresponds to a composite process in the Process Model. Consequently, the root of the Process Model corresponds to the root of the tree, the composite processes correspond to intermediary nodes in the tree while the atomic processes are represented as leaf nodes in the tree. Each type of node (e.g., `sequence` or `choice` and so on) employs a corresponding matching algorithm which verifies the compatibility between its IOs and the IOs of the query by taking into account the facts described before. The matchmaking algorithm begins at the root of the advertisement tree and recursively invokes the matchmaking algorithms for the roots of its subtrees finishing at the leaves. The process implies a sort of backtracking strategy in order to find the distribution of the request outputs over children processes. In such a way the algorithm tries all such possible distributions before returning a failure. For example, in the case of a `split` or `sequence` process, if the outputs requested by the query can be satisfied by all its children collectively then we have a success, otherwise a failure. Dually, if the

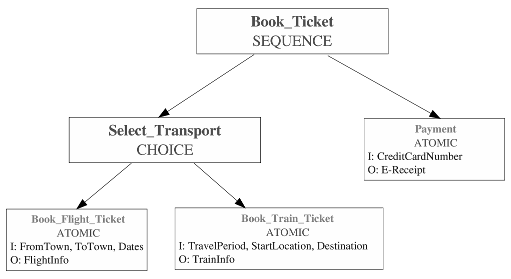


Figure 1: Process Model example of an advertisement.

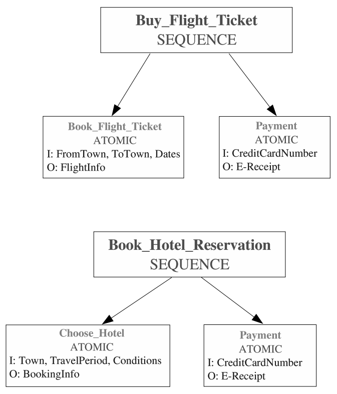


Figure 2: Example for illustrating limitations of previous approaches.

inputs requested by all its children can be provided by the request then we have a success, otherwise a failure. Or, in the case of a `choice` process we get a success or a failure depending on whether there exists at least one child able to provide by itself all the outputs desired by the query. Similarly, the success of matching the inputs depends on whether there exists at least one child whose inputs are provided by the query. Having said all this, by looking at the example in Figure 1 we can say that the query asking for inputs `CreditCardNumber`, `FromTown`, `ToTown` and `Dates` as well as `FlightInfo` as output leads to a success while the query asking for inputs `TravelPeriod`, `StartLocation`, `Destination` as well as `E-Receipt` as output leads to a failure as the request can not provide the input `CreditCardNumber` necessary for the execution of the `Payment` atomic process. A detailed description of the matching algorithms corresponding to several composite process can be found in [2].

Still, as presented in the Introduction, two of the main limitations of current state-of-the-art matchmaking algorithms are single service discovery and single service execution. The former relates to the fact that existing matchmaking strategies look inside the advertisements repository for a service able to fulfill the request by itself. Let us consider that the repository contains only the two services presented in Figure 2. For simplicity, let us assume further that there is an `exact` ontological relation between the `ToTown` and `Dates` inputs of the `Book_Flight_Ticket` atomic process and `Town` and `TravelPeriod` respectively, inputs of the `Choose_Hotel` atomic process. One can easily note that other matchmaking approaches confronted with a query specifying as inputs `TravelPeriod`, `FromTown`, `ToTown`, `Conditions` and `CreditCardNumber` as well as `Ticket` and `Reservation` as output give a `failed` match as there is no service in the registry providing both outputs. We have also assumed that `FlightInfo` and `BookingInfo` are sub-concepts of the `Ticket` and respectively `Reservation` output parameters desired by the query, in the ontology of the parameters. This leads to a `subsumes` match with respect to the output parameters. In the following section we will show how it is possible to tackle such limitation in order to obtain a match as for example it is obvious that by executing the two services in a `sequence`, the query can be satisfied. `Buy_Flight_Ticket` can be used in order to purchase the flight ticket while `Book_Hotel_Reservation` allows us to book a hotel room.

The second limitation we are addressing relates to the fact that existing approaches are concerned with matching the outputs of the request with the outputs obtained as a result of a single execution of the advertisements. Coming back to our previous examples and considering the advertisement presented in Figure 1 as well as a query looking for services which sell both train and flight tickets one can see that we get a `failed` match again. Still, the request can be fully satisfied by executing the advertisement twice as the `choice` process `Select_Transport` allows us to use either one of its atomic subprocesses in order to get both tickets.

3. PARTIAL MATCHMAKING

As mentioned before, the matchmaking algorithm of Bansal and Vidal returns a match only if there exists at least one service which satisfies the request alone as a result of a single execution. Such matched service should have its input set contained in the input set of the request and its output set should contain the output set of the request. It is obvious that this one-to-one match between the request and the advertisement is too restrictive and often fails as it is likely for the request to be fulfilled only by a composition of several advertised services. Let us consider the following example. Assume that in the service repository there are two services dealing with translations of documents among different languages. Let us further denote with S_1 the first service which translates a document written in German into the corresponding document written either in English, Italian, Russian, Spanish or French. With S_2 we shall denote the second service, slightly more complex than S_1, which either translates a document from Italian into Spanish, French, English or German, or it translates a document from Swedish into English, Spanish, German or French. Two possibilities for representing S_1 and S_2 as Process Models are shown in Figure 3 and Figure 4 respectively.

As one can note, S_1 consists of an `iterate` having a `choice` process as child. Each child of this `choice` is an `atomic` process which takes as input a document written in German and outputs the translation of this document into a different language (e.g., from German into Russian). Similarly, S_2 has an `iterate` process as root with a `choice` as child. This `choice` further splits into two other `choice` processes which contain atomic subprocesses needed for the translation of Italian and Swedish documents respectively into other languages. The `iterate` processes give the possibility to sequentially execute multiple translations.

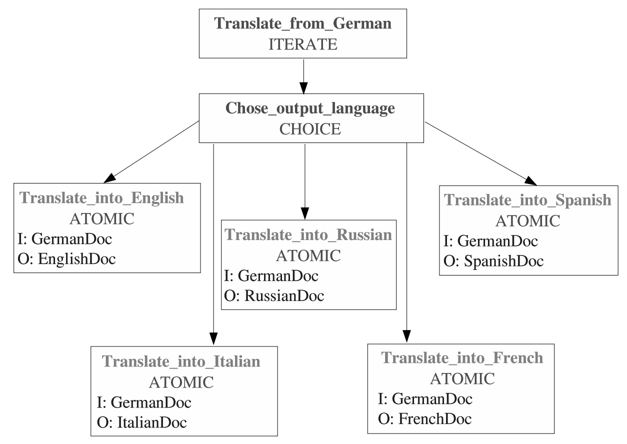


Figure 3: Process Model of service S_1.

Given these two advertisements, let us consider further a query Q expressing the desire of translating documents from Italian into Russian. Assuming that there is no other service in the repository beside S_1 and S_2, we can easily see that the matching algorithm proposed by Bansal and Vidal results in a `failed` match as there is no service S_k in the registry which can translate Italian documents into Russian. Yet, the query can be fulfilled. It suffices to execute S_2 so as to get a translation from Italian into German followed by the execution of S_1 so as to get the German document translated further into Russian. One can imagine several other scenarios in which the algorithm of Bansal and Vidal fails, for example given S_1 and S_2 as described above and a query Q asking for translations from Swedish and German into Spanish and French. As in our previous example multiple runs of S_1 and/or S_2 can satisfy the request.

The algorithm presented in this paper is intended to extend the work of Bansal and Vidal ([2, 3]) so as to enlarge the set of matched service advertisements. Its main idea comes from the fact that it is not always possible to find a service S_k in the repository which fulfills a query Q but a set $\{S_1, \ldots, S_i\}$ of services which composed in a certain way can collectively satisfy it. Our algorithm, as Bansal and Vidal, does not deal with the actual composition of services but its output is intended to be of help for the composition process.

In order to support partial matchmaking of services it is necessary to extend the concept of match with respect to previous works. As we mentioned before, a one-to-one match between a query and an advertised service occurs if and only if the inputs and outputs of the two entities follow certain rules. In the case of the algorithm proposed by Bansal and Vidal, the concept of match is associated to the entire service. Taking into account that atomic processes only have defined IOs in a Process Model as well as the fact that a particular run of a service can partially/fully satisfy the request, we introduce the notion of *flexible (partial) matchmaking.* We deal with the concept of match for atomic processes for which we define five levels of match:

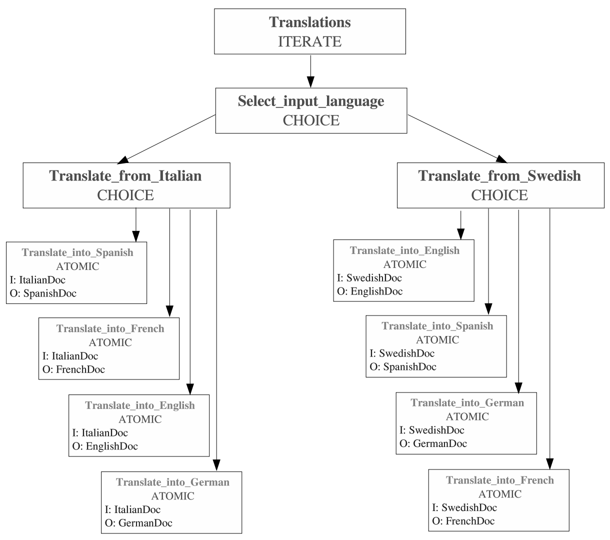


Figure 4: Process Model of service S_2.

0. *exact match*; all the inputs and all the outputs of the atomic process are contained within the request.
1. *partial match*; all the inputs and some -but not all- the outputs are contained within the request.
2. *inputs match*; all the inputs but no outputs are contained within the request.
3. *outputs match*; no inputs or some -but not all- inputs and all or some of the outputs are contained within the request. Finally,
4. *failed match*; no inputs or some -but not all- inputs and no outputs are contained within the request

The extension of the concept of match is necessary in order to individuate atomic processes which may prove to be useful for satisfying the request by properly aggregating (composing) their parent services. Going back to our example, if one issues a request wishing to translate documents from Italian into Russian, our partial matchmaking algorithm matches (among others) the atomic process `Translate_into_Russian` of S_1 at level three as well as the atomic process `Translate_into_German` of S_2 at level two. The `Translate_into_Russian` atomic process of service $S1$ is matched at level three because it generates one output `RussianDoc` which is requested by the query as an output as well, while its input `German_Doc` is not specified as an input in the query. Similarly, the `Translate_into_German` atomic subprocess of the `Translate_from_Italian choice` process of service $S1$ is matched at level two due to the fact that its input `Italian_Doc` is requested by the query as input as well, while its output `German_Doc` is not an output desired by the query.

Given a request, our flexible matchmaking algorithm analyses all the services in the registry. Each matched atomic process (at a level lower than four) is memorised into a data structure containing the following fields:

- `service` – the URI of the parent service.
- `atomicProcess` – the name of the atomic process.
- `parentType` – the type of the parent (composite) process (e.g., `choice`, `iterate`, and so on). Similarly to the algorithm of Bansal and Vidal, each DAML-S service is memorised as tree corresponding to its Process Model.
- `parentProcess` – the name of the parent process.
- `node` – the corresponding node in the tree.
- `levelMatch` – the level of match assigned by the algorithm (i.e., from 0 to 3).
- `degreeInputMatch` – the level of match of its inputs taking into account the ontological relations among them and the inputs desired by the query (i.e., `exact`, `plug-in`, `subsumes` and `failed`).
- `degreeOutputMatch` – similarly to `degreeInputMatch`.
- `matchSet` – a set of objects which make the correspondence between matched pairs of IOs of the atomic process and IOs of the request or outputs and inputs (OIs) of other atomic processes.

Our algorithm consists of four macro steps briefly described next:

1. *building DAML-S trees* – for each DAML-S service in the registry the algorithm builds the tree corresponding to its Process Model. The trees are memorised in a hash table using the service URI as key.
2. *matchmaking* – the matchmaking process cycles over the registry until all the outputs of the request have been matched or until no more matches can be found (i.e., no more data structures are added to the table).
3. *analysing the results* – the table obtained at the end of the previous step (*matchmaking*) is analysed so as to discard all the unnecessary atomic processes. Firstly, the atomic processes matched at level three are analysed. Due to the fact that not all of its inputs have been matched, such process can not be executed unless its remaining unmatched inputs are matched against outputs of other matched atomic processes (at levels 0, 1 or 2). Secondly, an atomic processes which is matched at level two is kept in the table only if at least one of its outputs matches at least one input of other matched atomic process (at levels 0, 1, 2 or 3). In this way the atomic processes matched at levels two or three are kept only if they have at least one output and all inputs respectively matching at least one input and outputs respectively of other matched atomic processes.
4. *output results* – the algorithm returns the overall degree of match, that is `exact`, `partial` or `failed`, depending on whether all the requirements of the request have been satisfied, or partially satisfied or totally unsatisfied. In the case of an `exact` or `partial` match, the algorithm also returns the table containing the matched atomic processes.

As just mentioned, the table containing detailed information on each matched atomic process is to be obtained at the end of the partial matchmaking algorithm. One may guess that some of the memorised data structures may not be useful with respect to the request, hence our algorithm analyses the table before returning it to the requester. In this way, atomic processes matched at levels two or three are discarded (as described by step three of the algorithm) given that they can not be part of any composition. Coming back to our example, if the request expresses the desires of having translations from Italian into Russian then the atomic processes of S_2 matched (during step two of the algorithm) at level two will be: `Translate_into_Spanish`, `Translate_into_French`, `Translate_into_English` as well as `Translate_into_German` which are children of the `Translate_from_Italian choice` process. Similarly, by analysing S_1 the algorithm matches `Translate_into_Russian` at level three. After checking the resulting table at step three of the algorithm, only `Translate_into_German` and `Translate_into_Russian` are kept, while the other matches are discarded. This is due to the fact that `Translate_into_Russian` previously matched at level three has its input `GermanDoc` matching the output of the `Translate_into_German` atomic process which has been matched at level two.

Our partial matchmaking algorithm returns one out of three possible answers constructed by confronting the query requirements with all of the services in the repository. The matching level refers to the satisfiability of the query requirements. It may be one of the following:

- *exact match* – all the requirements of the query are satisfiable. This means that one or more services have been found having their inputs contained in the inputs of the query and their outputs containing the outputs specified by the query.
- *partial match* – the requirements of the query are partially satisfiable. In other words, one or more services have been found having their inputs contained in the inputs of the query, but the outputs specified by the query are not entirely contained in the outputs provided by the matched services. The request has unmatched outputs.
- *failed match* – none of the requirements of the query are satisfiable. This means that there are no services in the repository which can provide by themselves or collectively (as a result of a composition) the outputs specified by the query.

As previously mentioned, additional to this output our algorithm returns the list of matched atomic processes. One might argue whether or not the partial match is of any use but we believe that it might prove to be useful as services come into and go out of the service registry. A partial match for a query can later change into an exact (or failed) one. Moreover, it is better to get a partial match instead of a failed one. Let us assume that the repository contains only services S_1 and S_2 as described by Figure 3 and Figure 4 respectively. If one issues a query requesting translations from Italian into Russian and Japanese, our algorithm responds with a `partial` match. As we have just seen, the child atomic process `Translate_into_German` of the `choice` process `Translate_from_Italian` of S_2 can be used together with the atomic process `Translate_into_Russian` of S_1 in order to have documents translated from Italian into Russian. Due to the fact that there is no advertised service which is capable of translating documents from Italian into Japanese, the query is partially satisfiable. Now, if we consider that at a later moment in time a new service advertisement (call it S_3) which offers the possibility of translating documents from Italian into Japanese is added to the registry, as well as by assuming that the query and its result have been previously cached, we get an `exact` match as the query becomes now satisfiable. Dually, if either one or more services are discarded from the repository we may get a `partial` match (e.g., *either* S_1 *or* S_2 *or both* are eliminated) or even a `failed` one (e.g., S_3 *and* S_1 or S_3 *and* S_2 are eliminated).

The core of our flexible matchmaking algorithm can roughly be sketched in pseudo-code as follows:

```
// Step #1: building DAML-S trees
init DAMLTable;
init listAPM;
for all i in registry.services do
  DAMLTable.put(URI(i), DAMLTree(i));

// Step #2: matchmaking
oldLen = -1;
newLen = 0;
while (!unmatchedOutputs.isEmpty())
      and
      (oldLen != newLen) do {
  oldLen = newLen;
  for all i in registry.services do {
    DAMLTree =
     GetRoot(BuildDAMLTree(DAMLTable.get(URI(i))));
    Match(DAMLTree);
    if unmatchedOutputs.isEmpty() then
      break;
  }
  newLen = listAPM.len();
}

/***
*    Step #3: analysing results. ‘‘OutputProcess012’’
*    refers to the output set gathered from all atomic
*    processes matched at levels 0, 1 and 2. Dually for
*    ‘‘InputProcess0123’’.
***/
for all i in listAPM do
  if LevelMatch(i) is three then
    if !OutputProcess012.contain(unmatched_inputs(i))
    then
      listAPM.delete(i);

for all i in listAPM do
  if LevelMatch(i) is two then
    if !InputProcess0123.contain(at_least_one_output(i))
    then
      listAPM.delete(i);

// Finally Step #4: output results
if unmatchedOutputs.size() >= requestedOutputs.size()
then
  print ‘‘failed match!’’;
else {
  if unmatchedOutputs.isEmpty() then
    print ‘‘exact match!’’;
  else
```

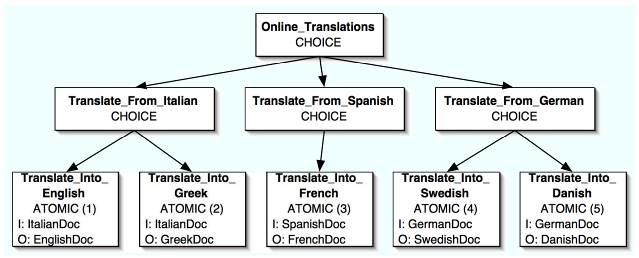

Online_Translations CHOICE
Translate_From_Italian CHOICE
Translate_From_Spanish CHOICE
Translate_From_German CHOICE
Translate_Into_ English ATOMIC (1) I: ItalianDoc O: EnglishDoc
Translate_Into_ Greek ATOMIC (2) I: ItalianDoc O: GreekDoc
Translate_Into_ French ATOMIC (3) I: SpanishDoc O: FrenchDoc
Translate_Into_ Swedish ATOMIC (4) I: GermanDoc O: SwedishDoc
Translate_Into_ Danish ATOMIC (5) I: GermanDoc O: DanishDoc

Figure 5: Advertisement no. 1

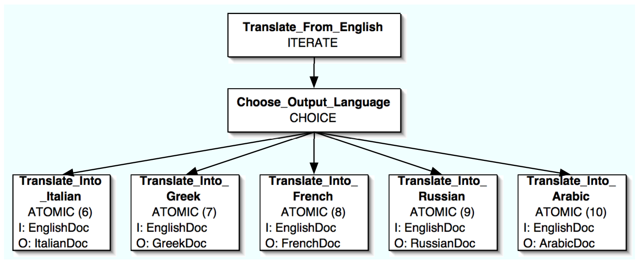

Translate_From_English ITERATE
Choose_Output_Language CHOICE
Translate_Into _Italian ATOMIC (6) I: EnglishDoc O: ItalianDoc
Translate_Into_ Greek ATOMIC (7) I: EnglishDoc O: GreekDoc
Translate_Into_ French ATOMIC (8) I: EnglishDoc O: FrenchDoc
Translate_Into_ Russian ATOMIC (9) I: EnglishDoc O: RussianDoc
Translate_Into_ Arabic ATOMIC (10) I: EnglishDoc O: ArabicDoc

Figure 6: Advertisement no. 2

```
    print ``partial match'';
  print DAMLTable;
}
```

The `Match` function begins its execution at the root of the advertisement tree and it is recursively invoked over the children nodes. The execution finishes at the leaf nodes, where the `Match` function verifies the compatibility between IOs of the corresponding atomic process and IOs of the query. To every type of DAML-S node (e.g. `sequence`, `choice` and so on) it corresponds a different `Match` function. In particular the `Match` function associated to atomic nodes determines and returns each atomic process' level by taking into account the classification described before. Moreover, the `Match` function inserts in the `listAPM` table all atomic processes matched at a level lower than four. The `listAPM` table is populated with each of the matched atomic processes at the end of the matchmaking phase when the `Match` function is invoked over all nodes of the the advertisement trees.

In order to give an evaluation of our algorithm let us consider the slightly more complex example in which the service registry only contains the three services shown in Figure 5, Figure 6 and Figure 7 respectively.

Figure 8 presents a comparison of the results obtained using the algorithm of Bansal and Vidal as well as ours for various queries. One can see that for the first query, the algorithm of Bansal and Vidal returns a `failed` match because there is no service in the service registry able to satisfy the query, that is to take as input at most an `ItalianDoc` and to give as output at least an `EnglishDoc` and a `GreekDoc`. Our algorithm answers with an `exact` match due to the matched atomic processes (1) and (2) of the first advertisement that are able to generate as output an `EnglishDoc` and a `GreekDoc` respectively. As an observation, the output given by our algorithm suggests that the first service should be executed twice in order to get both documents. The scenario is quite similar

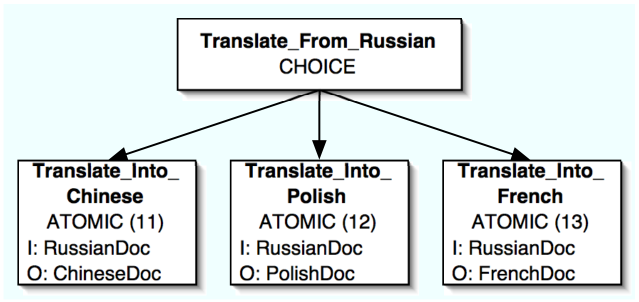
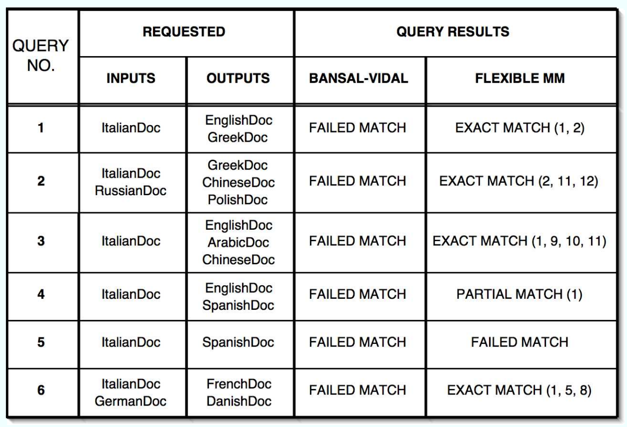

Translate_From_Russian CHOICE
Translate_Into_ Chinese ATOMIC (11) I: RussianDoc O: ChineseDoc
Translate_Into_ Polish ATOMIC (12) I: RussianDoc O: PolishDoc
Translate_Into_ French ATOMIC (13) I: RussianDoc O: FrenchDoc

Figure 7: Advertisement no. 3



QUERY NO.	REQUESTED		QUERY RESULTS	
	INPUTS	OUTPUTS	BANSAL-VIDAL	FLEXIBLE MM
1	ItalianDoc	EnglishDoc GreekDoc	FAILED MATCH	EXACT MATCH (1, 2)
2	ItalianDoc RussianDoc	GreekDoc ChineseDoc PolishDoc	FAILED MATCH	EXACT MATCH (2, 11, 12)
3	ItalianDoc	EnglishDoc ArabicDoc ChineseDoc	FAILED MATCH	EXACT MATCH (1, 9, 10, 11)
4	ItalianDoc	EnglishDoc SpanishDoc	FAILED MATCH	PARTIAL MATCH (1)
5	ItalianDoc	SpanishDoc	FAILED MATCH	FAILED MATCH
6	ItalianDoc GermanDoc	FrenchDoc DanishDoc	FAILED MATCH	EXACT MATCH (1, 5, 8)

Figure 8: Comparison with Bansal and Vidal

for queries number two, three and six. For the fourth query our algorithm answers with a `partial` match because the atomic process (1) of the first advertisement can be used in order to obtain the `EnglishDoc` from an `ItalianDoc` yet there is no advertisement that can give a `SpanishDoc` as output. Both algorithms respond with a `failed` match to query number five as there is(are) no document(s) in the registry able to take at most one `ItalianDoc` as input and to provide at least a `SpanishDoc` as output. One may note that the respective table summarizes the output generated by our algorithm.

4. CONCLUSIONS AND FUTURE WORK

Similarly to the work of Bansal and Vidal, we have implemented our algorithm in Java and we have used the DAML-JessKB package [9] for reading DAML-S service advertisements. By doing so, each advertisement is transformed into a (Subject-Verb-Object) triple memorised into the Jess knowledge base. Jess in also in charge of applying DAML rules. As described before, our algorithm builds a tree for each DAML-S service advertisement and given a query expressed as a set of desired IOs it returns the corresponding level of match as well as a table containing information about the matched atomic processes in the case of a match level other than `failed`.

We have seen how Web service matchmaking has evolved and has constantly improved during the last years: starting with (basic) keyword based electronic yellow pages (UDDI) to state-of-the-art semantic Web based techniques. From the latter ones, the most successful proved to be those based on ontology languages such as DAML-S. Matchmaking based on the DAML-S Service Profile only (e.g., [11]) was proven to have some limitations due to its lack of "understanding" of the inner structure (behaviour) of services, and as a result it did not lead to accurate results. A natural continuation recently came from Bansal and Vidal ([2, 3]) by building a matchmaking algorithm which uses the DAML-S Service Model so as to cope with the previously mentioned limitation. As we have argued in this paper, their algorithm is too restrictive in the way that it searches for a service advertisement only able to satisfy the query alone as a result of a single execution. As a natural extension, we took the matchmaking process one step further: from matchmaking done at the entire service (root process) level to matchmaking done at the atomic process level. By doing so, we allow queries to be matched not as in previous approaches on an one-to-one but on an one-to-many basis. A query can be matched by a set of service advertisements. In other words, given a query and a DAML-S service repository consisting of advertisements, our flexible matchmaking algorithm returns a positive match (i.e., `exact` or `partial`) if there exists a set of services in the repository having atomic processes whose IOs match query IOs or possibly other processes OIs such that the set of inputs of all the matched atomic processes are contained in the set of inputs provided by the query as well as the intersection between the set of outputs generated by the matched atomic processes and the set of outputs desired by the query is not void. The output generated by the algorithm consists of an `exact/partial/failed` match level as well as a table containing detailed information about the matched atomic processes which can prove to be useful in a composition capable of fully/partially satisfying the request. Such table is empty in the case of a `failed` match. One question one may ask is "why bother returning a partial match, knowing that the query can not be satisfied?". The answer comes from the fact that, in our view, services come into and go out from the repository, hence it is likely that a `partial` match can later change into an `exact` or `failed` one and vice-versa. One can simply think of a scenario in which a new service is advertised. Taking into account this newly added service, a query lacking an output at a previous run of the algorithm can now be fully satisfied by the services in the repository. The algorithm returns in this case an `exact` match. Dually, as services are discarded from the registry, it is possible to have queries that were previously fully/partially satisfied to be labeled now as `failed` matches due to the fact that their requirements can no longer be fulfilled by the existing services.

One direction of our future work points towards the investigation of techniques related to the field of Search Engines, such as scalability, spiders/crawlers, caching repository and queries as well as query indexing and so on, in order to optimize the matchmaking process.

Another direction of investigation is the automatic composition of Web services (e.g., [7]). Our algorithm makes a step in this direction by creating correspondences between atomic processes of different services having inputs/outputs matching other atomic processes outputs/inputs. One might note, however, that even in the case of an `exact` match with a set of services we do not have any guarantee that the matched services can actually be composed in order to satisfy the request.

Another interesting direction we intend to pursue is to further refine the matchmaking process through the introduction of a Service Type filter. Due to the semantic ambiguity of input/output parameters, one may get a positive match with a service which has nothing to do with the desire of the requester. We intend to tackle this issue by using an ontology of service types built similarly to the ones used for parameters and backed up by a corresponding matching algorithm.

5. REFERENCES

[1] R. Allen and D. Garlan. A formal basis for architectural connection. *ACM Transactions on Software Engineering and Methodology*, 6(3):213–249, July 1997.

[2] S. Bansal. *Matchmaking of Web Services Based on the DAML-S Service Model, Masters Thesis*. University of South Carolina, 2002.

[3] S. Bansal and J. Vidal. Matchmaking of Web Services Based on the DAML-S Service Model. *AAMAS, ACM*, pages 926–927, July 2003.

[4] A. Bracciali, A. Brogi, and C. Canal. A formal approach to component adaptation. *Journal of Systems and Software, 2004. In press.*, 2004. (A preliminary version was published in *Component Deployment*, First International IFIP/ACM Working Conference, LNCS 2370, pages 185-199, 2002. Springer.).

[5] H.-P. Company. Web Services Concepts – a technical overview. 2001. available at http://www.hpmiddleware.com/downloads/pdf ↱ ↱ /web_services_tech_overview.pdf.

[6] F. Curbera and et.al. Business Process Execution Language for Web Services (BPEL4WS). 2002. available at http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/.

[7] F. Curbera, R. Khalaf, N. Mukhi, S. Tai, and S. Weerawarana. The next step in Web services. *Communications of the ACM*, 46(10), October 2003.

[8] DAML-S Coalition. DAML-S 0.9 draft release. `http://www.daml.org/services/daml-s/0.9/`.

[9] J. Kopena and W. Regli. DAMLJessKB: A Tool for Reasoning with the Semantic Web. *IEEE Intelligent Systems*, 18(3):74–77, May/June 2003.

[10] L. Meredith and S. Bjorg. Contract and types. *CACM*, 46(10), 2003.

[11] M. Paolucci, T. Kawmura, T. Payne, and K. Sycara. Semantic Matchmaking of Web Services Capabilities. In *First International Semantic Web Conference*, 2002.

[12] M. Papazoglou. Service-Oriented Computing: Concepts, Characteristics and Directions. In *WISW2003*, pages 3–12, December 2003.

Distributed and Heterogeneous eHome Systems in Volatile Environments

Michael Kirchhof
Department of Computer Science III
Aachen University of Technology
Ahornstr. 55, 52074 Aachen, Germany
kirchhof@i3.informatik.rwth-aachen.de

ABSTRACT
eHome Systems consist of integrated and autonomous subsystems, which are installed on house owner's as well as on industry's side. Interaction is not limited to simple 1-to-1 relationships, they can cover arbitrary numbers of participants. When this is the case, it is expected that various middleware platforms come into play. In this paper a middleware called *Distributed Services Framework (DSF)* is proposed, which overcomes the limitations set up by J2EE, .NET, and similar ones. The proposed middleware is light-weight and flexible by incorporating existing middleware technologies instead of replacing them and provides an abstract and homogeneous view. The architecture of our approach is described and implications are analyzed. Furthermore, the proposed technology can be used for further development of Web Services technology.

Categories and Subject Descriptors
D.2.11 [**Software Engineering**]: Software Architectures; D.2.12 [**Software Engineering**]: Interoperability; D.2.13 [**Software Engineering**]: Reusable Software; H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Distributed systems*

General Terms
Design, Standardization, Languages

Keywords
DSF, Distributed Services Framework, Middleware, eHome

1. INTRODUCTION
As technology emerges to everyday life, it is brought to households, too. There is a variety of areas, in which the usage of technology (i.e., smart devices and computers) makes sense. These areas are shown in figure 1. Services of the first area (1) target the comfort of inhabitants, e.g., remote access and automatic control of appliances. Services in the security area (2) can be surveillance of the house or alarming the house owner if something unexpected happens. Communication services (3) comprise enhanced electronic mail and instant messaging. Services in the health care area (4) targets for example instructions in the case of illness or diets. Infotainment (5) stands for video on demand and similar. Consumption (6) targets the monitoring and optimization of energy consumption. Services in all these areas can be autonomous, i.e. the execution of them only depends on the computer system in the house. A natural extension would be the usage of remote systems providing information database and services. Services of this nature are also known as *value-added services*. Bringing an enhanced lifestyle to consumers and, in turn, bring new revenues to companies seems to open a new powerful market [35].

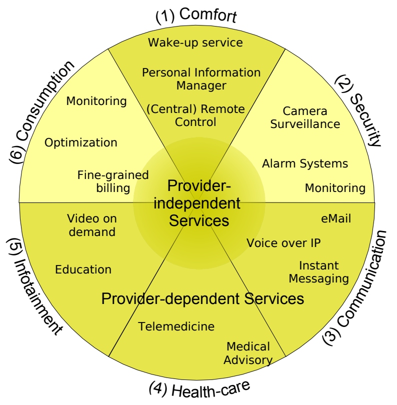


Figure 1: Areas of Services

In this paper we will take a look at systems combining automated homes, called *eHomes*, with enterprises and virtual enterprises. We call these systems *eHome Systems*. We will not focus on the insight of eHome Services, but on middleware technologies and the problems associated with heterogeneous systems. Talking about services, we mean any piece of software, which is executed in a *network environment*, making the usage and administration of ubiquitous appliances easier.

The scenario discussed in this paper is illustrated in figure 2. The connected home (1) at the bottom of the drawing is equipped with a so called *residential gateway* (2), a hardware device, which provides access to connection infrastructure and acts as runtime environment for the *service gateway*. The service gateway manages and runs eHome services. These services realize *eHome Services* (3), which are then visible to the house owner. Examples of this kind are an automated air conditioning system, or an automated energy control and optimization system. The household is equipped with several devices. These are connected with arbitrary network proto-

ICSOC 2004, New York, USA

cols to the residential gateway in the same manner as the services inter-communicate (4). The gateway acts as the central intelligence of the eHome. From the consumer's point of view, the residential gateway is a maintenance-free computing device. The connection (5) to the service provider is realized through a dialup connection (e.g., DSL or cable modem). The services interacting *sporadically* with the provider are connected via the Internet. User interaction (6) is realized via different kind of devices, e.g. mobile phones, PDAs, terminals, and personal computers.

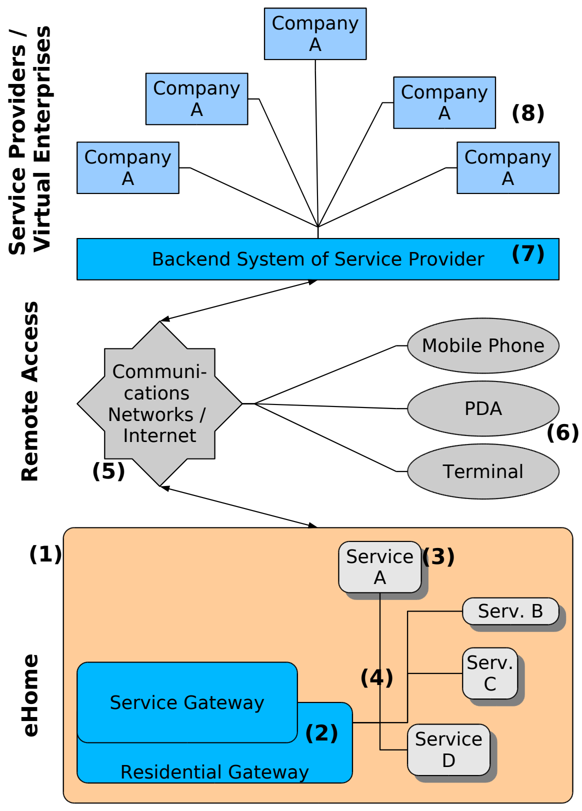


Figure 2: Scenario

One often mentioned problem is the existence of many established home automation standards. While several papers address this topic, we use an OSGi-compliant gateway [27, 13, 6] as the nerve center of our solution. The usage of the open service gateway enables an abstract, almost protocol-independent view onto the different home automation protocols used. Which protocols are commonly used will be described later. We are more interested in the mechanisms of *distributed services*, rather than in the mechanisms of *local communication and service development*.

We can think of several autonomous services brought to the house owner. But it has been shown that many services profit from knowledge base systems that are maintained by service providers and offer immaterial goods [2]. These providers do not get directly into contact with customers. They merely communicate with one dedicated company playing the role of the one and only service provider (7). Other companies (8) interact and provide services, management data, and specific databases. Together, they act as a virtual enterprise [21].

One application of these technologies is a modular alarm system. The alarm system consists of multiple cameras, multiple sensors, an outbound connection for alarm messages (e.g., email facility, SMS), and some power switches. All components are connected to the residential gateway. The residential gateway integrates and coordinates the components. The procedure is the following: When some of the sensors, for example motion detectors, detect something worth mentioning, a predefined subset of available switches (also called actors) are activated and selected cameras should take pictures of the location and store them. The house owner is informed about what is currently happening in his house in order to take adequate measures based on the kind of event and the pictures obtained from within his house. Possible actions would be to call the police in emergency cases or to discard the event, in situations where the cat raised the alarm accidentally.

With this setup, problems on three levels arise. On the first level the application logic is examined. The difficulty is, that it has to cover not only the service-oriented systems within the eHome, it has to cover the application logic on the remote site, and it has to deal with the communication and the interaction of both parties. All four dimensions of Enterprise Application Integration (EAI) can be found here: data integration, process integration, framework integration, and user interface integration [2]. The problems on the second level are associated with configurations of eHome Systems. Versioning and operating system differences are only two examples on this level when taking the large number of eHomes into account. The goal on this level is the automation of configuration and deployment. The third level deals with the integration of low-end devices, such as radiators or lamps. These do not have any kind of programmable intelligence, they merely implement one protocol and rarely more. While the scenario here is centralized, it is also possible to extend it with decentralized subsystems. There are research activities on these three levels at our department [31].

2. REQUIREMENTS AND ASSUMPTIONS

Home automation promises new comfort and useful services in our everyday life. These services [20, 23] will take place in the present day due to availability of ubiquitous appliances. From the user's point of view, convenience of service usage is the key factor. From the developer's point of view, different appliances and technologies exist, which have to be integrated. This imposes specific requirements on the development of services in the area of home automation.

Connecting home area networks with communication and data networks provides potential for many service ideas. Service remote configuration and maintenance are only two examples. Furthermore, users may access their eHome using different communication and data networks. Appliances can be controlled from anywhere (e.g., from the office) using a browser and the Internet. The owner of an eHome may determine and change the state of his alarm equipment with the Wireless Application Protocol (WAP [28]) and mobile communication networks using his mobile phone. Or, in case of an alarm, the alarm equipment sends a multimedia message (MMS [1]) to the owner of an eHome, who will receive the message with his mobile phone wherever he is. But eHome Services may also interact on their behalf with other data and communication network services. For example, these background processes could check automatically the integrity of the eHome and for updates/patches of the local components of installed eHome Services.

Developing eHome Services does not only mean to explore and realize consumers' needs and wishes. Realizing eHome services, *special requirements* have to be met:

- *eHome systems should be able to connect with different communication and data networks, supporting volatile and temporary connections.* The situation in eHome systems is that users are not expected to have permanent Internet connection. In fact, most eHomes will be equipped with dial-up connections which have to be triggered in a certain way. The triggering is a policy-based action. Several policies can be realized, if the starting point of the communication request is located within the local eHome. If the starting point is on a remote site (back-end system, Internet kiosk, etc.), the eHome has to be triggered to set up the Internet connection. This is crucial for successful eHome systems, because eHome users should be always in control of their systems and by that are responsible for the communication costs.

- *As new needs arise by users, new appliances or new services emerge, eHome systems should be* extensible *with services.* Not only the realization of new service ideas should be as easy as possible, but incorporating of existing services and a substitution of the underlying component-ware should be possible without destroying the system structure and the architecture of the complex application. Adding new services should also be possible without interrupting normal operation.

- *eHome Services need a way to communicate.* At present, the most common technologies for distributed object computing are Sun's J2EE Java Beans [9], OMG's CORBA [26], Microsoft's DCOM [5] and .NET, and Web Services [4, 14]. Current situation shows that software manufacturers make use of all these technologies, setting up a heterogeneous and fixed situation for each vendor, which prevents independently developed software to cooperate. To cope with that, either a new standard has to be accepted, which encompasses all existing technologies, or we need a modular and extensible system to integrate existing technologies. The latter would facilitate developing applications, which use modules implementing existing technologies to overcome the differences on syntactical and semantical level.

Building eHome services, we use components' functionality through the interfaces provided. Any component captures and hides details of its knowledge about a special device or technology within its implementation. Thus, a component-based and service-oriented approach for developing eHome services permits to concentrate on service logic, keeping details of technologies and protocols unconsidered. Every component framework provides a runtime environment for the components, respective services. The environment offers basic functionality like lookup, storage management, persistence, and an abstract access to operating system functionality. This *"glues"* the component to become an executable and accessible service. With this, it is possible to concentrate on the application logic and enables flexibility, safety, and reliability, which are the most important attributes for software systems [25]. Hence, we choose a component-oriented approach.

From today's point of view, the most feasible solution for back-end systems in eHome Systems is the Java 2 Enterprise Edition

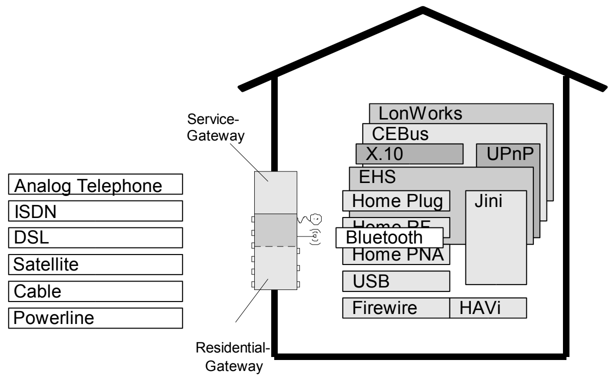


Figure 3: Home Networking Technologies

(J2EE) technology from SUN Microsystems [39]. The specification of such servers cover a broad range of the required attributes, but is in certain ways limited - volatile connections are not supported and the interaction with other services is limited to RMI and JMS technology. The situation for residential gateways is far more complex. An immense number of technologies have been developed. A few are shown in figure 3. One major problem is the interoperability of these technologies. To overcome this, we make use of an OSGi-compliant residential gateway. This gateway is specified by the Open Service Gateway Initiative (OSGi). It provides a runtime environment for component-oriented software called *bundles* and is able to integrate arbitrary home networking protocols. Main aspects of OSGi-gateways will be described in section 4.

The goal is to develop a communication framework, which allows easy and frictionless access to distributed and loosely coupled services over a volatile and temporary Internet connection. It should be light-weight and integrative on different platforms. Means have to be developed for service trading and remote usage. The framework should bridge the gap between the back-end systems of the provider and the eHome. This leads to a clear and integer view onto eHome Systems. *Summarizing, the gap is build up of the incompatible technologies for distributed object and service computing and the absence of a permanent Internet connection.* Furthermore, abstracting from concrete technologies would ease upgrading of services and even exchanging underlying technology. So the specific requirements are:

- remote service access
- transparent support for volatile and non-permanent Internet connection
- applicable for both back-end systems and residential gateways
- synchronous and asynchronous communication
- integration and interaction of arbitrary communication protocols
- access to protocol-specific attributes

In the following section, we will propose a solution which fulfills all of the above requirements.

3. THE DISTRIBUTED SERVICES FRAMEWORK

In this section we describe the main aspects of the distributed services framework (DSF). The work is influenced by Sun's J2EE, namely Enterprise Java Beans (EJBs). We chose this as our starting point, because EJBs are broadly used in production systems and there's a strong belief in this technology. With this predefinition, we can profit from the advantages of the Java programming language. As we will see, this does not obstruct the incorporation of other programming language domains.

3.1 Remote Service Access

Systems providing remote service access are usually built up from (1) the component framework, (2) services, (3) service users, and (4) the programming interface. The first two reside on the back-end system, the last two on the client system (i.e, the residential gateway). The programming interface is composed of classes for initializing and utilizing the remote component framework. The classes encapsulate the way how the interaction is realized. Figure 4 shows a coarse-grained system structure. Service components (i.e., EJBs) are plugged into the Enterprise Application Server (EAS) and made accessible by the framework. In order to request services, the programming interface of the EAS is used on the client side. The EAS programming interface hides technology details, here JNDI, the communication protocol, and the service reference. This means, that the client is bound to the programming interface of the remote system (i.e., of the EAS), as well as the interface of the service requested.

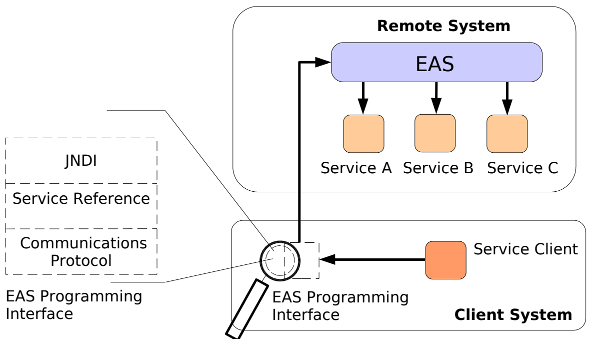


Figure 4: Remote Service Access in J2EE

As described in section 2, the hard-coded dependency of the remote framework has to be dismissed. This can be achieved by shifting the framework's programming interface to the server side. As the client can't directly contact the server without incorporating knowledge about the remote framework, a more general component is introduced at the client side, namely the *DSF programming interface* (see figure 5). To prevent the DSF programming interface to become as adhesive to the server framework as the J2EE programming interface, an opposite party has to be developed. This is the *DSF* component on the server side. It accepts requests from the DSF programming interface and interacts with the J2EE components on behalf of the client. In turn, the J2EE component handles interaction with the actually requested service.

eHome Systems will always be based on systems running a component-based framework (cp. section 2). With a symmetrical system structure, a bi-directional benefit could be achieved. Therefore, a DSF component is introduced at the client side (see figure 6 (a)).

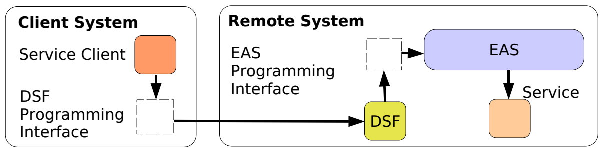


Figure 5: Remote Service Access with DSF

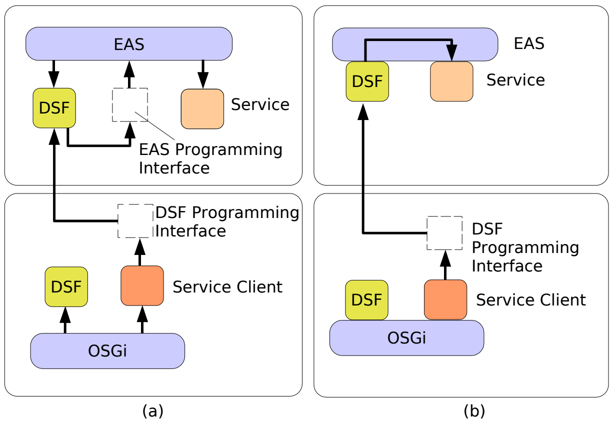

EAS
DSF
Service
EAS Programming Interface
DSF Programming Interface
DSF
Service Client
OSGi
(a)
EAS
DSF
Service
DSF Programming Interface
DSF
Service Client
OSGi
(b)

Figure 6: Component-based DSF

The framework related use-connections can be omitted, because they are inherent to the locally used framework (cp. figure 6 (b)). Only a service request initiated by the client and received by the server is depicted. A request from the back-end system to the gateway would be equally handled.

Now that the details of the component framework are hidden, the system can be extended to cut out implementation details of the used communication technologies. The details and the programming interface of the communication technology can be capsuled in components. We call them *connectors* and *acceptors*, because each communication requests has a sender and a receiver. If the communication is bidirectional, the roles can be switched. Such a situation is shown in figure 7: The DSF component does not directly contact the remote DSF component, it merely forwards the received request to the connector component where it is marshaled and transferred over the Internet. Then, the acceptor on the server side is connected, which unmarshals and forwards the request to the server's DSF component. There the translation to the programming interface of the EAS is done. Because of the symmetry, both directions are possible, allowing both unidirectional and bidirectional communication.

As described in the introduction, eHome Systems suffer from volatile and temporary Internet connections. This issue is addressed by the component *Connection Manager*. Its purpose is the encapsulation of the connection-related tasks. With this, it is possible to completely abstract from communication details, such as connecting and disconnecting to and from the Internet, request pooling, charging and policy-based connection management. Besides the Connection Manager, we introduce components for each communication technology (e.g., ISDN, DSL, Power Line). These components cover the protocols completely, keeping the Connection Manager simple.

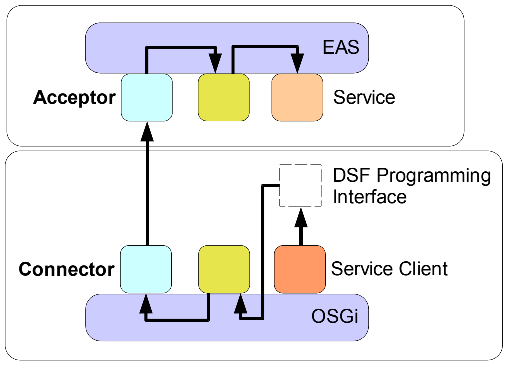


Figure 7: Connector Acceptor Concept

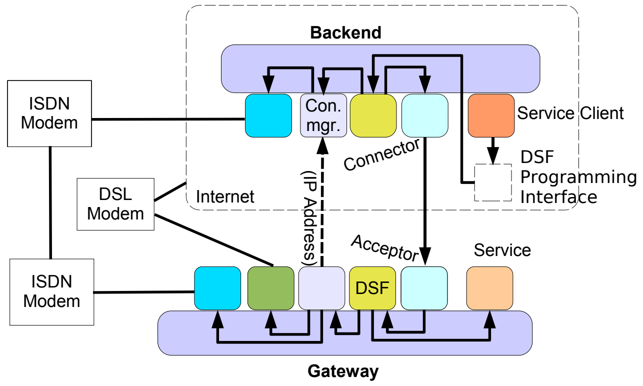


Figure 8: Distributed Services Framework (DSF)

The system structure is shown in figure 8. In the lower part, the *residential gateway* is shown. It is running an OSGi implementation. Bundles (cp. section 2) for *ISDN* and *DSL* are plugged in for connecting the Internet. The bundle *Connection Manager* depends on these and offers the postulated abstract view onto the connection to the remote site. The actual communication is encapsulated in the component *DSF*. This component communicates with *acceptors* and *connectors*, which handle the integration of the used communication protocols (e.g., HTTP, JMS, RMI). Besides these component, the actual services reside in bundles, too. The situation for the back-end system of a provider is nearly mirrored. This symmetry makes DSF more comprehensible. The only difference is, that the back-end side is not based on OSGi. Connections are handled by the *Connection Manager*. Core functionality is realized in the *DSF* component.

In figure 8 the situation is shown, where a service request is initiated by the back-end system. The service, which is actually emitting the request, uses the DSF programming interface to connect -either unidirectionally or bidirectionally- with a service residing on the residential gateway. DSF triggers the residential gateway to connect to the Internet in order to become accessible. This is necessary because the kind of Internet connection is property of the eHome user and usually the originator of a connection will be the payee. So, triggering must be free of charge. One possible solution is to use classified numbers for triggering the residential gateway, by comparing identifications of incoming calls. If the identification (i.e., the telephone number) matches, then the residential gateways connects to the Internet and becomes accessible for the back-end system.

Different communication technologies are integrated by the means of connectors and acceptors. For each communication technology an acceptor as well as a connector is required, while the complexity of these is low. One of the central issues is the transfer of service requests and the corresponding return values. The choice of communication technology is up to the DSF user (i.e., the service developer). The concrete technologies are encapsulated within the *connectors* and *acceptors*. Therewith, the system is open for upcoming technologies. Furthermore, the desired technology can be chosen according to specific attributes. Due to the fact that services are remotely accessed via proxies and proxies instances are tightly bound to service instances, we use the factory pattern [11] for connectors and acceptors. During communication, it has to be ensured that the connection between back-end and residential gateway (via the Internet) is kept. For this reason, both *Acceptor Service* and *Connector Service* are related to the *TemporaryConnectionManager*. So, dangling communications prevent the participating systems from disconnection. One problem was to assign asynchronous arriving results to the originator. It is solved by applying the *asynchronous completion token pattern* [33, 32].

3.2 Architecture

In this section, we will very briefly describe the architecture of selected components. In our understanding, the expressive power of the Unified Modeling Language Version 2 (UML2) [30, 34] is not sufficient for modeling service-oriented systems. So, we use an extended UML: (1) a software component denoted as a simple box, (2) services represented by a circle, (3) the relationship component implements a service described as a single line and (4) the use-relationship expressed by a directed single arrow. The use-relationship can incorporate arbitrary elements of the extended UML, i.e., classes, components, and services. This extension to UML2, where components and interfaces can be modeled, is necessary in order to make the important distinction between interfaces and services explicit.

With this extended UML, we can now transfer the abstract black-box view of connectors and acceptor into an architecture (see figure 9). As described, we make use of the factory pattern. The controlling functionality is located in the DSF component. Connectors are triggered and the DSF is triggered when acceptors receive data. Because of the non-permanent Internet connection, connectors and acceptors have to interact with the connection manager. To prevent the Internet connection to get cut off, the *Connection Blocker* component has to be accessed by the connectors and acceptors.

The Connection Management (see figure 10) can also be modeled with the extended UML. Starting from the situation with permanent connection to the Internet (shown on the right side), we specialized the architecture with respect to the requirements of volatile connections. With the decomposition of the system structure to the architecture of DSF, it became clear that the connection manager and the connectors/acceptors have to cooperate quite closely.

The services made accessible through DSF are a-priori unknown. So, it should be possible to incorporate services during runtime. This can be done using *Java Reflection* [36]. This leads to a problem with the Java runtime environment: In Java classes are loaded by the *Classloader*. It is required that the client-class and the in-

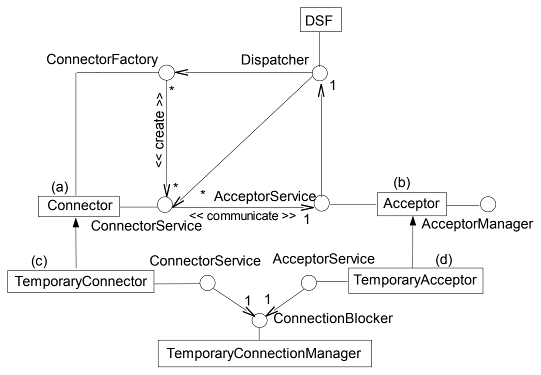


Figure 9: Connectors and Acceptors

terface of the requested classes are loaded by the same classloader. Otherwise the reflection mechanism will reject the request. Classloaders can only be shared, if classes or interfaces are exported from the same package. This is impossible, because the DSF component accesses per definition components *not* implemented within DSF. To solve this problem, the concept of *Service Adapters* was developed. These service adapters bridge the requests from DSF to the service, offering DSF a standardized interface to interact with eHome Services.

3.3 Implemented Prototype

The above described distributed services framework (DSF) has been realized as a prototype. In the current stage, connectors and acceptors have been developed for HTTP and JMS. HTTP was chosen because its wide use and its bidirectionality. Java Messaging Service (JMS) was chosen as an advanced unidirectional communication system.

The important part for a broad propagation of the DSF system is the programming interface. Figure 1 shows an example usage of the programming interface. In the first 4 lines the remote machine is identified. Then a handle to the DSF framework is gathered. The only thing left is shown in the 6th line: the remote service is identified, the protocol is specified, and the information about the remote site are added. After that, the remote service can be accessed by its own interface, which is made available through DSF.

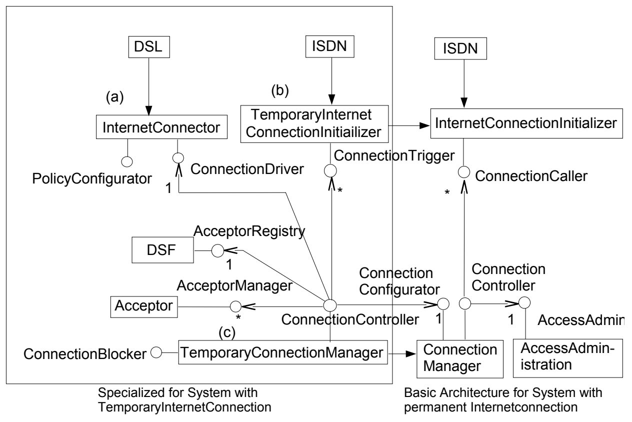


Figure 10: Connection Management

The similarity to Java Enterprise Beans' home and remote interface is desired. First, to give programmers an as least astonishing as possible means to work with DSF, and second, because the interfacing with EJBs is well understood and applicable to systems in the application domain of eHome Systems.

```
Properties props = new Properties();
props.setProperty( HttpServletProperties.HOST, "dukas" );
props.setProperty( HttpServletProperties.PORT, "8080" );
props.setProperty( HttpServletProperties.AUTHORIZATION,
                   "admin:secret" );

DSF dsf = new DSFImpl( bc );

MyService service = (MyService) dsf.create(
                                    "MyServiceAdapter",
                                    MyService.class,
                                    null, null,
                                    "http:",
                                    props );

/* Service Requests */

dsf.remove( service );
```

Listing 1: DSF Programming Interface

Based on the prototype, a web-based remote imaging system (PowerImage) and a multi-user Internet portal capable of triggering residential gateways has been developed. The portal stores user accounts in a database. These accounts contain triggering information among other information. When a user logs into the portal, he can choose to be connected to his eHome. For this purpose, either a trigger call (via telephone line) is issued or an IP-based signal can be emitted. The trigger signal is received by one ore more residential gateways. If the trigger code matches the ID of the residential gateway, the residential gateways sets up an Internet connection, which allows bi-directional communication. After this, the eHome owner can interact with his eHome, i.e. he can tweak settings of the installed eHome Services, start and stop services, and he can monitor his home. When the connection is no longer required, it can be closed either automatically or on demand.

4. RELATED WORK

Several approaches in the field of distributed service computing exist. In this section we will give an overview of these and point out the outstanding advantages of our solution. As we will show, none of the current developments offer a manageable and complete solution to the specific problems in the application domain of eHome Systems.

Solutions for Home Automation

In the area of home automation, several technologies have been proposed and established. One of the most notable is *OSGi* [27]. It targets the integration of different home automation standards (e.g., Jini, USB, EHS, EIB) and provides an execution environment for component-based services. The OSGi framework acts like a server and container for component-based services, which are denoted *bundles*. The figure shows some basic bundles for HTTP access (HTTP), logging (Log), and one communication protocol (USB). Other bundles are simply plugged into the framework, while the framework is the glue [38] for the component. A deeper insight into OSGi and its concepts can be found in [6].

OSGi enables friction-less access to different communication protocols and other services, but it is limited to local execution. For remote access and remote requests technologies have to be incorporated. Moreover, the temporary Internet connection imposes problems on development of distributed services based on a pure OSGi solution.

Another approach in the home automation area is *Connected Open Building Automation (COBA)* of the COBA Group [7, 8]. This group defines itself a standardization initiative. Their mission is to bundle the management of automated homes or automated industry facilities as much as possible. The proposed solution is based on an J2EE Application Server, incorporating different technologies in a multi-layer architecture. The target domain of this solution is the 1-point-management of facilities. On the one hand, remote access is not implemented nor specified. On the other hand, many important technologies can be integrated via COBA. Its shortcoming is, that distributed computing via volatile connections is not addressed.

Solutions for Distributed Computing

For distributed computing, several well established technologies exist. Prominent examples are CORBA [26], Enterprise Java Beans [9], COM+/DCOM/.NET [24, 29, 22, 17], and RMI [15, 37]. A promisingly approach is the Java Messaging Service (JMS) [16, 12]. Actually it is a messaging system, but can be used for initiating actions on a remote site. A homogeneous solution relying on only one of the earlier mentioned technologies would solve the problem of distributed computing, while the temporary connection is still an issue. Only solutions based on JMS do not suffer from the volatile connection. The problem with these technologies is that none of them has been put through. This may be caused by different qualities of the technologies and their aptitude in certain domains. Consequently, we don't expect any convergence in this field. This requires an integrative solution, like the one proposed by us.

Web Services Technology [41, 3, 14] comprises a component-based approach to describe, publish, localize, and invoke distributed services. So, Web Services allow the interaction of distributed applications via the Internet. This is achieved by standardized web protocols, i.e. XML, WSDL, UDDI, and SOAP. This has certain merits and demerits: On the one hand, the approach is simple enough to be widely adapted. On the other hand, the constriction to SOAP forces the adaption of all components, which should be integrated, while the reasons for choosing a specific distribution and communication technology are discarded. Furthermore, the non-permanent connection to the Internet is not addressed. Several proposal for a broadening of the Web Services Standards exist, mainly focusing on separation of concerns [40], asynchronous communication and transactions [42], and composition and collaboration of Web Services [43, 10]. Beside the distribution of interfaces and interoperability in more or less homogeneous systems, none of the other relevant requirements in the context of eHome systems are met.

5. SUMMARY AND OUTLOOK

In this paper we have discussed the special requirements of eHome Systems - complex distributed systems based on diverse technologies and a non-permanent Internet connection. We introduced the *Distributed Services Framework*, which can overcome the problems of integrating arbitrary communication protocols. First and foremost, it fits very well into an OSGi-based residential gateway and second it has a clear and straight-forward programming interface. This eases the introduction of DSF *and* insures the investment in back-end systems and the large number of residential gateways in the intended mass-market. Due to the applicability in back-end-to-eHome situations and in inter-provider situations, it provides means for continuous development and realization of eHome systems.

The concept has been proved with a prototype developed at our department. The implemented functionality has shown, that the usage of DSF makes the interaction between residential gateways and back-end systems very easy to implement and to maintain. In the current status, connections can be established over HTTP and JMS, each prominent representatives of synchronous and asynchronous communication. Support for HTTPS (for secure communication) and SOAP is planned for future releases. Furthermore, buffered and policy-controlled transfer is discussed in our group. This could be applied for example for accounting data and other data not requiring real-time transfer. One important feature would be the specification of upper bound of delay, ensuring the transfer at all.

Open problems can be seen for not-serializable data and bulk data useful for Infotainment eHome Services. eHome Systems are systems for the mass market. Likewise, synchronous administration of several residential gateways has to be solved. Scalability and security issues are to be observed. Also, ensuring that the system is available and failsafe is an open problem. Other areas for future work are the problems of a data layer covering the distribution aspects while addressing security issues. The development of eHome Services at a higher level, i.e. the composition of services, is observed, too.

The separation of concerns is preserved throughout the whole system. One very important attribute of the DSF framework is, that it is *not* a self-contained middleware platform like J2EE and Web Services. It is merely wrapping established and coming middleware platforms. This puts developers into a position, from where the middleware and back-end system technology choice is easier, because of the capability of DSF to change -even during runtime- the underlying technology.

We do feel confident, that the proposed framework provides a flexible and extensible solution to the problems in distributed eHome Systems. Several eHome-services (described in [19]) are currently ported to DSF to further validate the proposed approach. The discussed aspects can be used to enhance Web Service technology and to broaden the applicability of Web Service technology.

6. ACKNOWLEDGEMENTS

This work has been part of a cooperation between RWE AG, Essen (Germany) and the Department of Computer Science III at RWTH Aachen University of Technology (Germany). Mr. Jens-Michael Peters (RWE AG, Essen, Germany) played an important role with his insight of the eHome market and Dr. Elmar Pritsch (Booz, Allen & Hamilton, Düsseldorf, Germany) in discussions about requirements and business cases for eHome Systems.

7. REFERENCES

[1] 3GPP. Multimedia Messaging Service, TS 22.140. `http://www.3gpp.org`, 2002.

[2] S. Becker, M. Kirchhof, M. Nagl, and A. Schleicher. EAI, Web und eBusiness: Echte Anwendungsintegration macht Aufwand! In *Proceedings of Online '02*, Congress VI, pages C630.01–C630.27, 2002.

[3] J. Bloomberg. Web Services and a new Approach to Software Engineering. *The Rational Edge*, Apr. 2002.

[4] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard. Web Services Architecture. Technical report, World Wide Web Consortium (W3C), Feb. 2004. `http://www.w3.org/TR/ws-arch/` (7.5.2004).

[5] N. Brown and C. Kindel. *Distributed Component Object Model Protocol DCOM*. Microsoft Corporation, 1998.

[6] K. Chen and L. Gong. *Programming Open Service Gateways with Java Embedded Server Technology*. Sun Microsystems, 2001.

[7] COBA Group. COBA Tutorial. Technical White Paper, COBA Group, Nov. 2001. `http://www.coba-group.com/files/COBA_tutorial_en.pdf` (13.5.2004).

[8] COBA Group. A Framework for COBA Server. Technical white paper, COBA Group, Nov. 2002. `http://www.coba-group.com/files/COBA_WP_011119.pdf` (13.5.2004).

[9] L. DeMichiel, L. Yalcinalp, and S. Krishnan. *Enterprise Java Beans Specification, Version 2.0*, 2001. Sun Microsystems, Inc.

[10] V. Ermolayev, N. Keberle, and S. Plaskin. Towards Agent-Based Rational Service Composition - RACING Approach. In Jeckle and Zhang [18], pages 167–182.

[11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[12] P. Giotta, S. Grant, and M. Kovacs. *Professional JMS*. Peer Information Inc., Mar. 2001. ISBN 1-861-00493-1.

[13] L. Gong. A Software Architecture for Open Service Gateways. *IEEE Internet Computing*, 5:64–70, Jan. 2001.

[14] K. D. Gottschalk, S. Graham, H. Kreger, and J. Snell. Introduction to Web Services Architecture. *IBM Systems Journal*, 41(2):170–177, 2002.

[15] W. Grosso. *Java RMI. Designing and Building Distributed Applications*. O'Reilly, Oct. 2001. ISBN 1-565-92452-5.

[16] M. Hapner, R. Burridge, R. Sharma, J. Fialli, and K. Stout. *Java Message Service*, Apr. 2002. `http://java.sun.com/products/jms` (3.12.2003).

[17] D. Iseminger, editor. *COM+ Developer's Reference Library*. Microsoft Press, 2000.

[18] M. Jeckle and L.-J. Zhang, editors. *Web Services - ICWS-Europe 2003, International Conference ICWS-Europe 2003, Erfurt, Germany, September 23-24, 2003, Proceedings*, volume 2853 of *Lecture Notes in Computer Science*. Springer, 2003.

[19] M. Kirchhof and S. Linz. Component-based Development of Web-enabled eHome Services. In *Proceedings of Ubiquitous Mobile Information and Collaboration Systems Workshop 2004 (UMICS 2004)*, Lecture Notes in Computer Science. Springer, 2004. to appear.

[20] E. Kovacs. *Vermittlung und Management von Diensten in offenen Systemen*. Verlag Dr. Kovac, 1999. ISBN 3-540-63518-1.

[21] L. M. C. Matos, editor. *Collaborative Business Ecosystems and Virtual Enterprises* , volume 213 of *IFIP International Federation for Information Processing*. Kluwer Academic, 2002.

[22] J. Löwy. *COM and .NET Component Services. Mastering COM+ Services*. O'Reilly, Sept. 2001.

[23] M. Merz, K. Müller, and W. Lamersdorf. Service Trading and Mediation in Distributed Computing Systems. In *Proceedings of the 14^{th} International Conference on Distributed Computing Systems*, pages 450–457. IEEE Computer Society Press, 1994. Poznan (Polen).

[24] Microsoft Corporation, editor. *Microsoft .NET Framework 1.1 Class Library Reference*. Microsoft Press, Apr. 2003.

[25] M. Nagl. *Softwaretechnik: Methodisches Programmieren im Großen*. Springer, Heidelberg, 1990. ISBN 3-540-52705-2.

[26] Object Management Group, Inc. *The Common Object Request Broker: Architecture and Specification, Revision 2.6.1*, 2002. `http://www.omg.org` (14.6.2002).

[27] Open Services Gateway Initiative. OSGi Service Platform. `http://www.osgi.org` (13.11.2003), October 2001. Release 2.

[28] Refsnes Data. WAP/WML Tutorial. `http://www.w3schools.com/wap/`.

[29] J. Richter. *Applied Microsoft .NET Framework Programming*. Microsoft Press, Jan. 2002.

[30] J. Rumbaugh, I. Jacobson, and G. Booch. *Unified Modelling Language Reference Manual*. Addison Wesley Longman, Feb. 1999. ISBN 0-2013-0998-X.

[31] RWTH Aachen University of Technology, Department of Computer Science III. eHome Group. `http://www-i3.informatik.rwth-aachen.de/ehome`.

[32] D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann. *Pattern-orientierte Software-Architektur. Muster für nebenläufige und vernetzte Objekte*. dpunkt.verlag, 2002. ISBN 3898641422.

[33] D. C. Schmidt. A family of design patterns for applications-level gateways. *Theory and Practice of Object Systems*, 2(1):15–30, 1996.

[34] K. Scott. *Fast Track UML 2.0*. a! Press, 2004. ISBN 1-590-59320-0.

[35] Sun Microsystems. The Connected Home. `http://www.sun.com`, 2002.

[36] Sun Microsystems, Inc. *Java Core Reflection API and Specification*. JavaSoft, 1997.

[37] Sun Microsystems, Inc. Java Remote Method Invocation (RMI) Specification, 2004. `http://java.sun.com/j2se/1.5.0/docs/guide/rmi/spec/rmiTOC.html` (03.6.2004).

[38] C. Szyperski. *Component Software*. Addison-Wesley/ACM Press, 2 edition, 2002. ISBN 0-201-74572-0.

[39] A. Thomas. Enterprise JavaBeans Technology - Server Component Model for the Java Platform. Technical report, Patricia Seybold Group, Boston MA, Dec. 1998. prepared for Sun Microsystems.

[40] B. Verheecke, M. A. Cibrán, and V. Jonckers. AOP for Dynamic Configuration and Management of Web Services. In Jeckle and Zhang [18], pages 137–151.

[41] World Wide Web Consortium (W3C). Web Service Activity. `http://www.w3c.org/2002/ws`, 2002.

[42] U. Zdun, M. Völter, and M. Kircher. Design and Implementation of an Asynchronous Invocation Framework for Web Services. In Jeckle and Zhang [18], pages 64–78.

[43] L.-J. Zhang and M. Jeckle. The Next Big Thing: Web Services Collaboration. In Jeckle and Zhang [18], pages 1–10.