

**Oracle Corporation**

Village d'Entreprise Green Side DANIEL SERAIN  
400 Av. Roumanille, BP309 Mobile +33 607333346  
06906 Sophia Antipolis Cedex Fax +33 4 93008844  
France

---

# **The Ever Increasing Power of Middleware**

---

Essay on the concept of service and its impact on  
Middleware Business

Dr. Daniel SERAIN  
*Presales Fellow*  
*Oracle EMEA Middleware Technology*

---

## Table of Contents

<b>Table of Contents</b> .....	<b>1</b>
<b>Table of Figures</b> .....	<b>2</b>
<b>Executive Summary</b> .....	<b>3</b>
<b>Business view</b> .....	<b>4</b>
<b>The service lifecycle</b> .....	<b>5</b>
Develop/describe phase.....	5
Publish/discover phase .....	7
Execute/interact phase .....	8
<b>Summary of current state</b> .....	<b>8</b>
<b>What needs to be done to roll the stone up?</b> .....	<b>10</b>
<b>What would be the benefits?</b> .....	<b>11</b>
Wish #1: Availability of a uniform programming service model .....	11
Wish #2: Ability to map service dependencies .....	12
Wish #3: Unambiguous data description.....	12
Wish #4: Unambiguous service description.....	14
Wish #5: Formalization of the non-functional attributes of a service (ex: QoS).....	16
<b>Conclusions</b> .....	<b>17</b>
Consequences on the service model .....	17
Some general comments .....	20
Consequences on SOA middleware business .....	20
General conclusion .....	22

---

## Table of Figures

<i>Figure 1: Representation of a Service</i> .....	5
<i>Figure 2: Key drivers for SOA Governance initiatives</i> .....	9
<i>Figure 3: Greatest challenges in managing change in SOA</i> .....	10
<i>Figure 5: The service model including SCA</i> .....	12
<i>Figure 6: Example of UDEF indexing</i> .....	13
<i>Figure 7: Service interoperability using UDEF keys</i> .....	13
<i>Figure 9: Service model including SCA and UDEF keys</i> .....	14
<i>Figure 10: Service Ontology</i> .....	15
<i>Figure 11: The resulting service model</i> .....	18
<i>Figure 12: Comprehensive Service Description</i> .....	19
<i>Figure 13: A possible timeline for the introduction of new technologies in the IT business World</i> .....	22

## Executive Summary

This essay is about understanding the evolution of infrastructure middleware, this software layer sitting on top of the application server and under the applications with the main purpose of transferring data between applications' components. In the early 90's, this layer contained only few communication protocols such as Corba<sup>1</sup> for objects, queuing for applications, or RPC<sup>2</sup> for user interface software. Today, this layer has grown significantly and its market size is in billion of dollars (\$4.4B in 2008, for IDC<sup>19</sup>). Functionality wise, it provides interoperability between software services and as a result, is tightly linked to the evolution of the service entity. Understanding the evolution of the service concept should help us understand how infrastructure middleware will evolve.

Many activities are deployed around the concept of Service. They go in many directions but they all have one thing in common: they contribute to enrich the service entity. Hence, the service entity is expandable, but how? To answer this question, a service model is introduced. It is made of two parts: the descriptive part and the code module part seen as a black box. The descriptive part provides information about the black box part and is used and interpreted by the middleware. Today, all the innovation is done on the descriptive part, which immediately translates into more functionality in the infrastructure middleware.

Several domains will be explored to show this evolution. The first one is about Service Component Architecture (SCA), which goal is to facilitate distributed applications development. Its study shows the move from standard programming (module part) to descriptive programming to automate service interaction. This technology will be announced at Oracle Open World 2008 and will be part of the SOA Suite release 11.

The second domain and most important one is about semantics. Up to now, service interoperability has been described at the syntactic level. The next major phase of Internet middleware is likely to be in the semantic domain. It will happen along three dimensions: data semantics to ensure that the exchanged data is well understood by both parties, service semantics to ease service interoperability and allow automatic service composition, and finally Quality of Service (QoS), a domain which is becoming more and more complex. Semantics introduces reasoning techniques, which are required to draw conclusions from incomplete and imprecise data. The impact of semantics on the service model will be shown as well as the consequences on the infrastructure middleware components and tools.

It is difficult at this stage to get quantitative data from analysts on the impact of those new technologies on SOA middleware business, because it will take several years before they appear on the market. SCA will be the earliest, about a year from now. Semantics technologies are at least three years away. By that time, the SOA software market<sup>21</sup> will have reached \$14B and will be matured with 77% of companies using the service approach. For Y. Natis<sup>22</sup> from Gartner, it will be time for intelligence within services.

SCA and semantics show that Infrastructure middleware is going to dramatically evolved in the coming years. A wealth of new technologies incorporating intelligence will be added to this middleware layer. This is a new wave, which, most likely will be the semantic wave.

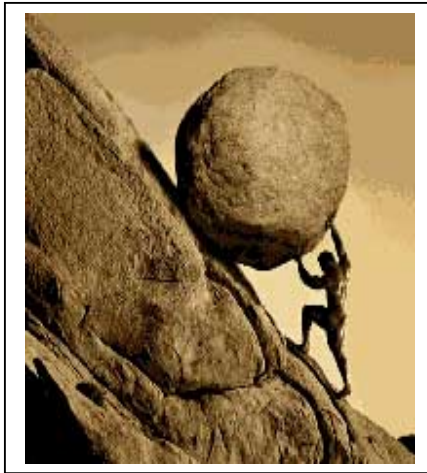
---

<sup>1</sup> Corba: Common Object Request Broker Architecture. Standard communication protocol between objects

<sup>2</sup> RPC: Remote Procedure Call

## Business view

The myth of Sisyphus could certainly apply to software. As a human being, Sisyphus wanted to achieve what only gods could have, that is eternity. For a while, the very clever Sisyphus was able to put Death in chains. Zeus realizing that nobody was dying anymore sent Ares to deliver Death. As a punishment for challenging gods, Sisyphus was condemned to roll, for eternity, a rock on top of a hill, each time the rock falling back before reaching the top.



It seems that with software human beings behave like Sisyphus. Each new technology wave carries the promise to reach the top of the hill where all IT environments will run smoothly and will quickly respond to business needs. However, is that the IT final goal? Rather, it seems that the top of the hill is a moving target. IT makes constant progress toward a changing goal. In the process, problems are solved and progress is made. In this paper, the IT technology wave is called SOA, Service Oriented Architecture, and its goal has been mentioned above. The objective of this paper is to propose some new steps, which should help Sisyphus in his inhuman task.

Today, how far is SOA on his way to reaching the top of the hill? If we refer to a study done by the Aberdeen group<sup>3</sup> on 400 companies, 100% of them recognize a cost reduction in development, 72% a cost reduction in maintenance, 61% estimate that SOA improved the quality of service to the end user and 60% think that their IT system is more agile. Thus, SOA seems to provide already positive results but has not yet reached its full potential. Let us identify the impediments in order to resolve them and go further up.

In SOA, an infrastructure called middleware is necessary to run the services. The more sophisticated services are, the more complex is the infrastructure. The goal here is to show the likely evolution of services required to address the new business challenges, and evaluate the resulting impact on the infrastructure. This last point is important for two reasons: first, it is perceived that productivity in IT will come from middleware and second, because all new technologies added to the middleware layer represent new markets and new opportunities.

In SOA, the fundamental component is the Service. It is the building block of an IT solution to a business need. The concept of Service needs to be clarified. Here, we shall use the following definition from Alex Maclinovsky<sup>4</sup>:

*An **Enterprise Service** is a self-contained component delivering business functionality combined with an extendible set of non-functional, policy-driven qualities (such as security,*

---

<sup>3</sup> "SOA Middleware Takes the Lead: Picking Up Where Web Services Leaves Off", Aug 7<sup>th</sup>, 2007

<sup>4</sup> "Quest for true SOA", published in InfoQ

*industry/customer defined service policies, management, monitoring, and lifecycle management) which responds to requests through a well-defined, standard, published interface.*

Based on this definition, it appears that a service is made of two independent parts: the Service Module that contains the code and, the descriptive part called the Service Interface (see Figure 1). Today, the descriptive part is limited to the description of the interface of the service (how to communicate with it), i.e. the WSDL<sup>5</sup> file. The focus of this essay is on this descriptive part because our belief is that it will support most of the growth of the service model.

*A service is made of two independent parts: one part contains the code and the other describes how to use the code.*

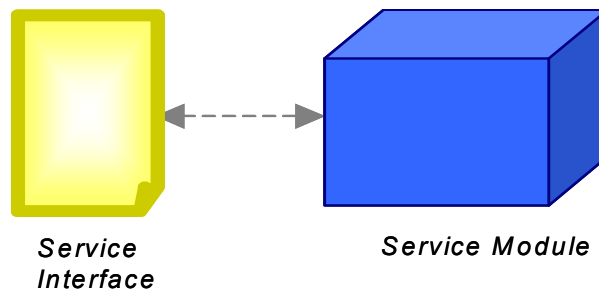


Figure 1: Representation of a Service

To study the evolution of Services, we shall look at their lifecycle (develop/describe, publish/discover, execute/interact), analyze the limitations of the current model with respect to business needs, identify the characteristics of new technologies and size the business benefits.

## **The service lifecycle**

Services are made of two parts, the service module part, which contains the code of the service, and the interface, which is in a separate file containing a description of the service. The creation of a service implies to *develop* the code (the algorithmic part) and to *describe how to use it* in the interface. The interface describes the interaction of the service with the external world. Its content is interpreted by the middleware to generate appropriate actions. This paper demonstrates that the evolution of the services imposed by the new business requirements, is likely to take place on the descriptive part of the service. More information will be added to the interface and appropriate processing will be needed to interpret it. Those new functionalities will belong to the middleware infrastructure.

## **Develop/describe phase**

Today, assuming we know exactly which functionality is required, services can be built either from scratch, or by reusing one or several existing pieces of code. Let us consider both cases.

*Service to be entirely coded.* If the needed web service requires to be entirely developed, some decisions need to be taken about the selection of the programming language (Java, BPEL, .Net) and about the interface. With web services, the interface is written in WSDL

---

<sup>5</sup> WSDL: Web Service Description Language

and the communication protocol is SOAP<sup>6</sup>. At the interface level, a key activity is to define unambiguously the inputs and outputs parameters accepted and generated by this new service. This task should not be neglected because studies have shown that today, 35 to 40% of the cost of building service interoperability<sup>7</sup> (two services exchanging data) comes from making sure that the data exchanged as input and output parameters are semantically equivalent. To reduce this cost, a formal description about the type and nature of data needs to be provided. This is not done today. Then, when the WSDL interface is completed, it is published in a repository for later discovery and reuse.

*Service reused.* Reusability has always been stressed as a key feature of SOA. However, after several years of implementation of SOA solutions, some figures are becoming available. Statistics coming from IBM<sup>8</sup> and from Crédit Suisse<sup>9</sup> show that between 5 to 10% of the business services are being reused. The figures are much higher for lower level services such as data services. To try to understand why, let us see how reusability is currently achieved. Several situations can occur:

- First case, the code comes from a legacy application that we wish to turn into a set of web services. A question we may ask is: is it worthy to do it? For IBM, the answer is yes. IBM estimates that it is five times less expensive to reuse existing applications than to write new code. In the SOA customer projects that they delivered, they estimate that 43% of the total effort was spent on reusing existing applications. 50% was devoted to implement new functionalities. Gartner<sup>10</sup> has done a study that shows that when a new business application needs to be developed within a large firm, 70% of its functionalities already exist among internal applications. This statistic can be refined further by IBM's data that states that only 20% of the functionalities available in an existing application have the potential to be reused. The combination of both statistics leads to a level of reuse of the application code of 14%. It is interesting to understand that these statistics are about reusability of existing applications, not of existing services. This distinction is important because the level of reuse is very different: the reused of application code (14%) is about twice the level of reused of business services (5 to 10%). It looks as if when the application code is transformed into web services, its level of reuse goes down. This certainly reflects the current trend of transforming existing applications as a set of web services.

- Second case, the web service already exists and is published in a repository. We discuss in the next paragraph the issues associated to discovering an existing service.

- A third possibility is the design of a new service obtained from combining existing services. This is the case for instance when the new service is written in BPEL, which structures the flow between other services. To build such a service, the programmer will need to know the BPEL technology, to discover and understand the interfaces of the various services, and to activate them. Today, reusing such a composite service is not easy because it does not exist as a unique entity but as a set of independent services<sup>11</sup>. Each composing service is stored independently with no external references to the other services and with no formal description of their connection structure. However, some approaches exist to simplify it. For instance, Oracle has introduced AIA, the Application Integration Architecture. It is helpful since it provides complete integration solutions and a

*Reusability of services is very low. Business service reusability is between 5 to 10%*

---

<sup>6</sup> SOAP: Simple Object Access Protocol

<sup>7</sup> Ron Schuldt, Open Group presentation, Glasgow 2008

<sup>8</sup> Data given by IBM at the Open Group conference, Paris, April 2007

<sup>9</sup> Web Services conference, Gartner, Europe 2007

<sup>10</sup> Massimo Pezzini, Gartner European Web Services conference, Rome, June 2007

<sup>11</sup> A BPEL service is one service. In the current SOA environment, if this service calls four other services, that makes five entities to manipulate and deploy.

development environment. But, it does not address the general problem of interoperability and reuse, it just provides already made, very focus solutions.

### **Publish/discover phase**

Once services are developed, they need to be stored in a place where we can search for them, understand what they do, how they perform, what is needed to activate them, what are their various releases and their usage policies. These tasks are called publishing and discovery. Two types of storage are available to publish services, the service registry and the service repository. The service registry is a catalogue of services (some time called the yellow pages) that helps in service definition, service selection and in enforcing service policies. The service repository consists of various artifacts/assets about the services including functional specifications, authorized users, documentation and various other service artifacts including SLAs that define transaction capacity, maximum throughput, downtime etc. Service registry and service repository are key to governance. While service registry is normally used to accommodate run-time assets, service repository is used both for design time and run-time assets.

When a developer gets a request for a given service, he needs to have a precise specification of it. Using this specification, he must first find out if such a service already exists. He does that by searching in the service registry/repository which may contain hundred of services. If taxonomy is used to classify services, he needs to figure out in which part of the taxonomy the service should belong to. Service naming and description being not standardized, the programmer has to understand the meaning of the parameters of the selected services to make sure the service does what is expected.

The repository can also be accessed at runtime for dynamic look up and binding. Today, such a dynamic service discovery and invocation can be implemented in Java (not within Oracle BPEL PM or Oracle ESB). Assuming dynamic discovery is achieved, then remain the problem of understanding the interface, the input and output parameters. How does the program know what those parameters mean and what the service does? There is clearly a need for semantics to provide a clear meaning to data, and an engine to reason about it. Today, there is another pattern, which offer an intermediate solution and which can be used in Oracle BPEL PM and Oracle ESB: it is the dynamic invocation/partner link. This means that the programmer knows the service interface and semantics (what the service is supposed to do), so he understands input and output data as well. At runtime the system can dynamically select a service implementation to be called (e.g. a communication channel service to publish messages over a channel. For every channel there will be a different implementation, but always the same interface and behavior). Then it is possible to dynamically decide, which implementation to call.

In reality, specifications of services today are not well formalized and programmers tend to be discouraged when looking for a service. As a result, they tend to redevelop the service since anyway, it is their code that they love the best and trust the most. The same story happened with Object technology. Reuse was very low and many redundancies existed.

An important parameter to service selection is the notion of non-functional attributes of a service (ex: Quality of Service). It may happen that several services exist, providing similar functionalities but having different characteristics in terms of concurrent users, response time, security, cost, etc. Those elements should be formalized to contribute to the selection of the most appropriate service. Today, there is no way to automate this

*Services lack formal description of their parameters, functionality, and non-functional attributes*

process because of a lack of formal representation to describe the non-functional characteristics of a service.

## **Execute/interact phase**

Today, in a business scenario implementation, the flow of execution of services has to be predetermined. It is programmed at design time using the BPEL language and is contained in a business service. This is the case for instance for an order management application in which the full process is described in BPEL and at each step a service is called, using data provided by the previous service. This approach is powerful but imposes to have every step coded at programming time. There is no support for a dynamic definition of the flow based on real time data or external event, and on non-functional characteristics of the services. Let us consider two examples.

Automatic web service discovery: For example, the user may want to find a service that sells airline tickets between two given cities and accepts children under 12 years old travelling alone. Currently, a human must perform this task. He might use a search engine to find the proper web page and determine if the web site satisfies his constraints. This process can be automated if web sites offer web services functionalities and publish them. However, it is not enough. For a search engine to dynamically locate web services that can provide a particular class of service capabilities, while adhering to some client-specified constraints, much more information than syntactic description of the services is needed as we shall see it later in this paper.

*Automatic service discovery and composition require a new set of technologies*

Automatic web service composition and interoperability: Let us consider the case of a man who wants to offer a nice week-end to his wife. He would like to fly (no more than 2 hours) to a sunny place (at least 23° during day time). They want to play golf, spend a night in a five star hotel, and his budget is 2500€. In this case the web is seen as able to reason about information, called appropriate services, in order to provide a solution. This is the semantic web vision. The best way to write such a service is to use rules-based programming, a technique introduced in artificial intelligence systems, back to the 70s. The flow of execution is data driven, which means that the order in which the services are called is not known in advance. This program will have to discover, select and activate airline services, hotel services, weather services, and possibly golf services. Those services will be combined to offer a higher level service. The purpose of this example is to show a new way of using the web that represents a new class of problems that cannot be entirely addressed by today available technologies.

In the business world, when a company needs to select and use Services, very quickly the problem of quality of service (QoS) emerges. This leads the service client and the service provider to agree on what should be delivered and under which conditions. This is formalized in what is called a Service Level Agreement (SLA). Today in SOA, there is no formal way to express at the service level the QoS parameters. This is a pity because in a B2B or B2C environment, various offerings can provide similar service functionalities that will differ for instance, on cost, time for delivery, or reliability. Without those non-functional attributes, an SOA system cannot take into account, at runtime, users' requirements.

*Non-functional service parameters are as important as functional parameters*

## **Summary of current state**

This brief overview of the service lifecycle puts in the light some problems. It shows that the development environment is made of a heterogeneous set of software (Web services, EJBs, .Net components, legacy applications, etc) that the programmer tries to unify by using WSDL and SOAP, which is not always the most appropriate communication protocol. Some workaround technologies exist such as WSIF (the Web Services

Invocation Framework), to deal with heterogeneous environments. WSIF is a simple Java API for invoking Web services, no matter how or where the services are provided. However, WSIF does not provide a generic solution.

Another issue comes from the different nature of services: some are used locally (e.g. data services) and require fast access, as opposed to business services that are more global and have the potential to be shared across the net. These later services need to be externally available and as such require supporting the SOAP protocol. Thus, two different types of services are emerging, based on their usage that can either be local or global.

The development process shows the wide expertise required by the programmer. Besides the fact that the programmer needs to understand the business problem, he needs to be able to program in languages such as Java, BPEL, WSDL, and needs to know communication protocols such as RMI, IIOP, and SOAP. Knowledge of other technologies such as WSIF is recommended. Obviously, programmer's time cannot be entirely devoted to the business problem.

Reuse of composite services is very hard to achieve because of a lack of a model to represent composition of services. Today, only written documents are available to describe such services<sup>12</sup>.

Efficient service discovery is a way to speed up service development, improve service reusability, and allow dynamic service search. This is not the case today because the functional and non-functional characteristics of a service are not described in a formal way. Since no ontology is used, no reasoning can be applied, and as a result, no powerful search engine can be built to locate a given service if it exists in a repository. The availability of this type of technology would lead to automatic composition of services.

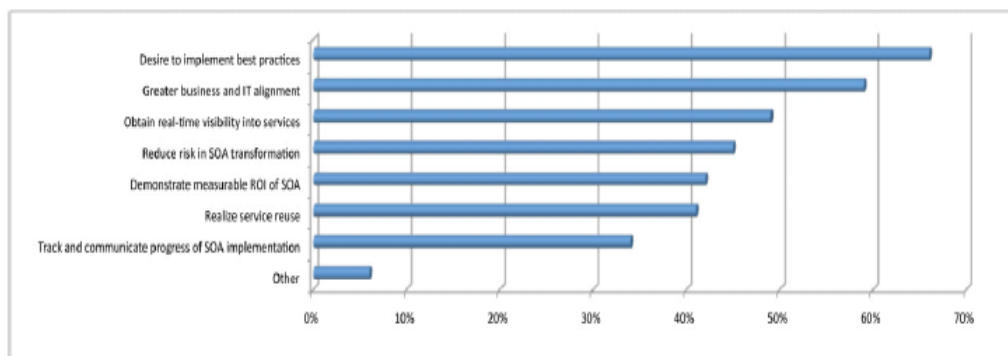


Figure 2: Key drivers for SOA Governance initiatives

At runtime, only static, predefined business processes can be implemented. As it was shown, not all the problems can be addressed that way. There is also a lack of support for handling non-functional attributes of services such as QoS.

---

<sup>12</sup> The WS-Choreography standard focuses on the flow of exchange of messages between services, the sequence and conditions in which the messages are exchanged. The underlying idea is that a choreography will become a reusable pattern. A side effect of the Choreography is that the relationship of the services is known, but it is not its main purpose and it does not make it explicit and easy to exploit. It should be noted that this technology appeared at the same time than BPEL and has not reached the same level of use and success.

A last issue, but certainly not the least, is the poor description of the inputs/outputs parameters used by the services. Today, very little (if none) documentation is provided. This results in an increase of the cost of interoperability by a factor of 35 to 40% because the developer has to understand the details of the target service to identify the proper data. This activity could be quite complex and time consuming if the target service is part of an ERP such as SAP. Today, there is no way to automate this process, because of a lack of formalized knowledge.

At this stage, it may be interesting to consider the study done by ebizQ<sup>13</sup>, summarized in Figure 2. This study is about SOA governance and lists the key drivers. It is interesting to see that “Service reuse” among a list of eight drivers, comes number six. Thus, reusability that has always been stressed as a key feature of SOA is going down in the list. “Obtain real-time visibility into services” is a more important driver than reuse. “Greater business and IT alignment” remains a key motivation.

Two other findings of this ebizQ’s survey (see Figure 3) are related to what companies expect from SOA governance. They are:

1. Being able to trace the relationships and dependencies that connect services to each other.
2. Traceability of the connection between services, business objects and business functionality.

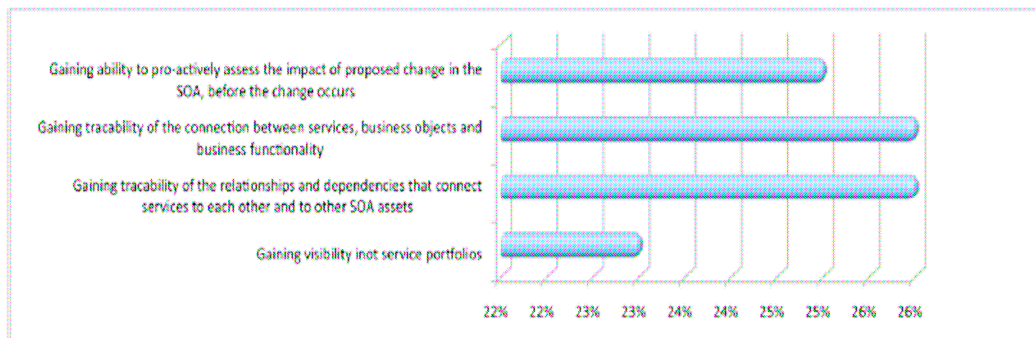


Figure 3: Greatest challenges in managing change in SOA

Companies want to quickly analyze the impact of changes made to individual services. It is critical for organizations to have ways of analyzing the impact of change (or potential change) and of understanding how alterations to individual services will affect the rest of the environment. That is where impact analysis and traceability come in. In fact, in the eyes of the users, adaptability is a key aspect of SOA.

*SOA is about adaptability and reusability*

### **What needs to be done to roll the stone up?**

Today, to move the rock on top of the hill, Sisyphus’s engine (the service) runs on one cylinder only (the WSDL interface). To get the full power of a four-cylinder engine, new technologies need to be added. To build such an engine, Sisyphus has made a set of wishes:

<sup>13</sup> ebizQ’s survey entitled “Increasing the Effectiveness & Efficiency of SOA Through Governance”, 2008

- *Wish #1: Availability of a uniform programming service model*
- *Wish #2: Ability to map service dependencies*
- *Wish #3: Unambiguous data description*
- *Wish #4: Unambiguous service description*
- *Wish #5: Formalization of the non-functional attributes of a service (ex: QoS)*

To roll the rock on top of the hill, Sisyphus has been looking for help and heard about new technologies, not all well tested and mature, but just coming out of research labs or standard committees. He thinks that they could help him in his Dantesque task. One is called Service Component Architecture (SCA) that could equip him with a unique programming service model. But, this is not enough. Sisyphus believes that only knowledge and intelligence introduced into IT technologies would really make the difference. His last hope is about using Uniform Data Element Framework (UDEF) and Ontology Web Language (OWL). Those two technologies refer to semantics. Only knowledge about data and services and the ability to reason can help him achieve the goals described above. In this chapter, only their main characteristics are provided when needed.

*Sisyphus believes that brute force is too limited and that only intelligent technologies able to understand the environment and adapt in real time will save him from his eternal task*

## **What would be the benefits?**

### **Wish #1: Availability of a uniform programming service model<sup>14</sup>**

**Description:** Such a model will hide the heterogeneous nature of the software components and of their interfaces. It should facilitate reuse. The uniform model should allow describing how services are combined (ex: composite services) and it should let the infrastructure middleware decide which communication technology is the most appropriate to address the local or global use of the service. Programmers should not have to worry about those technologies.

**How can it be achieved?** Service Component Architecture (SCA) defines a simple, service-based model for construction, assembly and deployment of a network of services (existing and new ones) that are defined in a language-neutral way. This model allows the creation of components (providing the services) and more importantly offers a mechanism for describing how those components work together. From the point of view taken in this essay, the interesting aspects of the SCA approach are:

1. It provides a unique programming interface to components that supports and combines various technologies. Components can be built in Java, BPEL or other languages. The common assembly model will specify how those components can be combined to provide a new application. It introduces the notion of *Composite* that contains a set of components providing a specific business function that can be reused.
2. It provides a clear separation between the internal fine-grained services that are targeted for local reused and the business coarse-grained services that have the potential to be externally reused and are exposed to the outside world.

---

<sup>14</sup> This programming model does not exist today. Implementation techniques solve the issues, for instance with the service bus that provides appropriate interfaces to the various service types but at the programmer level remains the task to use the various communication protocols.

3. The description of the solution model is written in a configuration file, called the Service Component Description File (SCDL) that complements the WSDL file. This description file will be handled by the SCA middleware runtime.

**Business Benefits:** SCA saves time and money by simplifying system development. It increases programmer productivity (shorter learning curve with fewer errors) and improves system efficiency. SCA enables and encourages reuse through the notion of *composite*. It simplifies the workload of developers and architects, freeing them to work on things that add value to the business.

**Impact on the service model:** the service model needs to be complemented by the SCDL file (see Figure 4).

SCA provides in the SCDL file the connectivity structure between services

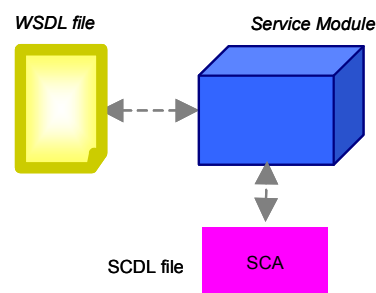


Figure 4: The service model including SCA

### **Wish #2: Ability to map service dependencies**

**Description:** Relationship between services should be clearly expressed in the description of each service.

**How can it be achieved?** The SCA model associates to each component a *reference* attribute that describes its dependency on services provided by other components. To perform its task, a component may rely on other services. This dependency is expressed by using *references*.

**Benefits:** There are threefold. First, when using a service the programmer knows exactly which other services will be needed. Second, when activating a service, the middleware will know in advance which other services will be called. It can start activating them to improve system performance. Third, the knowledge of the relationships between services provides the traceability required to analyze the impact of change on one service. Adaptability of the system will be increased as requested by users (see ebizQ's survey).

### **Wish #3: Unambiguous data description**

**Description:** When a data is used, there should be a method to clearly understand what this data means. Meta data provides the meaning of a data. A *date* is a piece of data, but a *person's birth date* is information. The target method should support automation and should uniquely define the data.

**How can it be achieved?** The ISO/IEC 11179 standard provides a way to unambiguous describe a data element. It associates to the data element what it calls a *data element*

concept that describes it unambiguously. The Open Group that has created the Universal Data Element Framework (UDEF) has used this approach. It provides a naming system with associated unique identifiers for indexing and aligning semantically equivalent concepts. Each data is fully characterized by a UDEF key. UDEF is a fundamental tool for semantic interoperability and information management.

UDEF associates to each data element a unique key

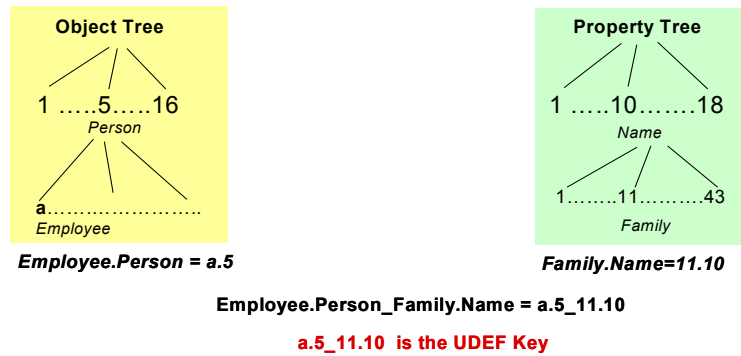


Figure 5: Example of UDEF indexing

The UDEF indexing system is based on the ISO/IEC 11179 recommendation for representing metadata for an organization in a Metadata Registry. It uses two concepts: *Object* and *Property*, which are developed into a tree of predefined terms with a number, associated to each tree element. Those two trees have been generated to cover the domain of *Enterprise*. As an example (see Figure 5), the Family Name of an Employee will have as a UDEF key: *a.5\_11.10*.

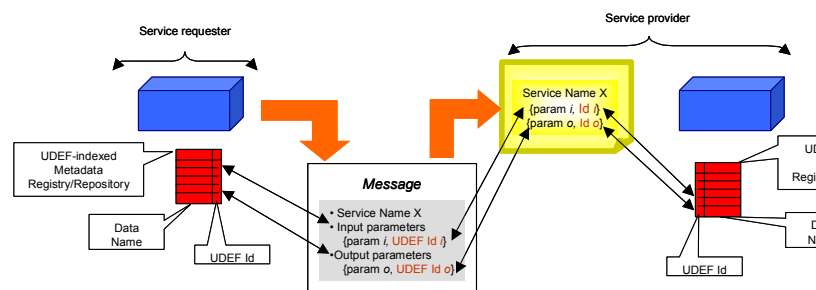


Figure 6: Service interoperability using UDEF keys

**Benefits:** By providing semantic interoperability, key business benefits can be achieved in using UDEF. In an SOA environment, interoperability between services will be simplified, errors will be avoided, and development cost will be reduced by 35 to 40%<sup>7</sup>.

The introduction of the UDEF keys when describing the interface of a service solves the interpretability problem of the service parameters. In the same way, standard message formats such as ebXML, EDI, or EDIFACT to name a few, should also be UDEF indexed. The mapping between message data and service data done by the service adapters will be greatly simplified. The adoption of UDEF will be accelerated if actions were taken in

those standard committee to map messages' parameters with UDEF keys. Figure 6 shows the communication between two services using UDEF keys.

The concept of role in the context of UDEF, needs to be clarified. Let us consider two data elements such as the *Employee entity* and the *Traveler entity*. They will have two different UDEF keys. It is likely that a traveling service will have as one of its parameters a *Traveler entity*. In the case of an Employee travelling, it will be necessary to map the *Employee entity* to the *Traveler Entity* parameter. This means that the mapping to the Service parameters is not always between identical UDEF keys. Some intelligence is required when doing the mapping. UDEF is there to clearly understand the data element but it does not address the concept of role. To be honest, it is not its purpose.

In the business intelligence domain, when data consolidation is needed, using UDEF key ensures that the system is using the correct data. The same is true in data mining and data warehousing where it is important not to mix apple and oranges.

Impact on the service model: UDEF keys need to be added to the WSDL file. This can be represented as follows (see Figure 7):

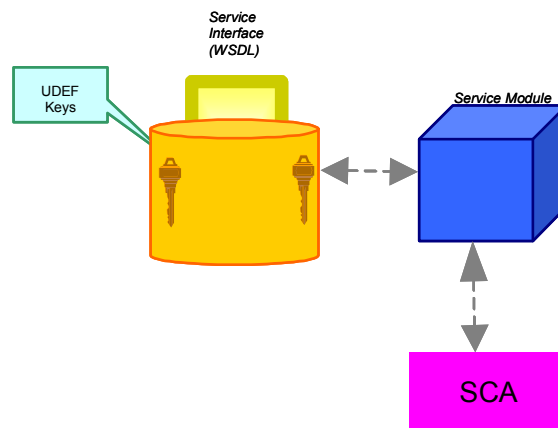


Figure 7: Service model including SCA and UDEF keys

#### **Wish #4: Unambiguous service description**

**Description:** In the business world, when a customer wants to buy a service from a supplier, he needs to describe in details the expected functionalities that he is looking for (ex: For a “Generate pay slip” service, there is a need for input parameters, outputs, pre-conditions, delivery date, etc). The same should be expected from software services. It should be possible to express WHAT the service does, HOW, with which data, under which set of conditions. This represents a mandatory step toward automatic service discovery and composition. Today, the only available formalized description of a service is given by the WSDL interface that provides syntactic information about the name of the service, its parameters and the mapping with the message format. Our goal is to build a system which answers the WHAT and HOW questions about a service.

How can it be achieved? The approach taken here is to use a language based on Ontology. Ontology is about the exact description of things and their relationships. For the web, ontology is about the exact description of web information and relationships between web information. A key aspect of ontology is that it can be used to formulate queries. With this in mind, the Ontology Web Language (OWL) was introduced. An extension to this

language has been defined to specially deal with service semantics, it is called the OWL-S, the Semantic Mark-up Language for Web Services.

The Semantic Web should enable greater access not only to content but also to services on the Web. Users and software agents should be able to discover, invoke, compose, and monitor Web resources offering particular services and having particular properties, and should be able to do so with a high degree of automation if desired. Powerful tools should be enabled by service descriptions. OWL-S is an ontology of services that makes these functionalities possible.

OWL-S has three main parts (see Figure 8):

1. The *service profile* to advertise and discover services: The service profile tells, "*What the service does*", in a way that is suitable for a service-seeking agent to determine whether the service meets its needs or not. This form of representation includes a description of what is accomplished by the service, limitations on service applicability and quality of service, and requirements that the service requester must satisfy to use the service successfully.

OWL-S provides the means to identify a service

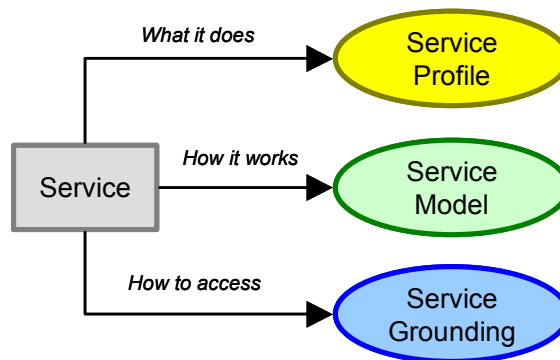


Figure 8: Service Ontology

2. The *service (or process) model*, which gives a detailed description of a service's operation. The service model tells a client how to use the service by detailing the semantic content of requests, the conditions under which particular outcomes will occur, and, where necessary, the step by step processes leading to those outcomes. That is, it describes how to ask for the service and what happens when the service is carried out.

3. The *grounding*, which provides details on how to interoperate with a service, via messages. Typically, *grounding* will specify a communication protocol and message formats. Inputs and outputs parameters are perfectly described as long as we remain within the OWL-S world. At this level, it is important to note that, when service grounding takes place, a mapping needs to be performed to the messaging model that can be anything (ebXML, EDIFACT, RosettaNet, etc). To ensure that the mapping can be properly achieved, the best way is to associate a UDEF key to every type (or subclasses) of parameters.

Input and Output Service parameters should have a UDEF-key to map them to the external world of OWL-S

**Benefits:** By combining OWL-S and UDEF, services are completely defined and uniquely identified. As a result, they can be automatically located, in a local service repository or on the web at development time and at run time. This will improve services reuse, avoid duplication, and reduce development time. It is also a necessary step for automatic service composition.

**Wish #5: Formalization of the non-functional attributes of a service (ex: QoS).**

Description: In the business world, when a customer needs to select a service (for instance a daily delivery of lunch food to a company of 200 employee) from a set of offers, very quickly the problem of quality of service (QoS) emerges. QoS is very often a decisive criterion for service selection. When a service is considered, the service client and the service provider need to agree on what should be delivered and under which conditions. This is formalized in what is called a Service Level Agreement (SLA). Today, in the software world, selection of services (by a user or by a software agent) is based on functionality only. This approach is not sustainable when considering selecting services from the entire web, and possibly in a dynamic way. When searching for a particular service, the client should be able to express what he expects from the service, functionally but also non-functionally, and the service provider should describe precisely what the service does and how. Referring to the Lunch food service, the customer may want a three courses lunch, with bread and equal balance of fish and meat as the main course. As a non-functional parameter he expects the main dish to be delivered warm. On his side, the food producer would expect that ovens would be available at customer site to warm up food. In order to match specifications from both sides a common language should be used to express them.

How can it be achieved? A language is required to describe both the client's requirements and the provider's service characteristics. Among other things, the language should describe the set of service level indicators (QoS parameters) and their associated metrics to allow measurement. It should also indicate the set of level objectives (SLOs) to be fulfilled by each one of those indicators. It should be noted that QoS parameters refer to observable properties relating to non-functional aspects of the service, e.g. availability, performance and reliability. Then, reasoning can be applied to identify the best offer.

An SLA language assumes the existence of a QoS/SLAs ontology. QoS semantics allows the user to define service quality requirements. Providers publish the QoS of the services being offered using this ontology. The fact that ontology is used allows to apply rules and to perform reasoning. The key aspect here is the ability to reason about the available information. This task is performed by a broker which can do an automatic selection of the best service offer (ex: in terms of quality and price) among those made available by providers.

Many efforts have been undertaken concerning the representation of quality of service concepts<sup>15</sup>. Some of them are toward the definition of specific languages (ex: WSLA<sup>16</sup> or WS-Agreement<sup>17</sup>) to present agreements including quality of service. Others are centered on the definition of a QoS ontology to provide a machine-understandable semantic representation of QoS information. Much ontology exists and an initiative has started to create a unified ontology.

*Ontology in OWL-S allows a broker to reason about QoS information and to find the best match.*

---

<sup>15</sup> Several standards exist to represent non-functional parameters. OWL-S provides a framework in which it is possible to use whatever technology/standard which is most appropriate and on which there is most agreement. Currently, to represent non-functional parameters OWL-S suggests using an ontology but it is not mandatory. The Web Services Policy Framework (WS-Policy) provides a general purpose model and corresponding syntax to describe the policies of a Web Service. If it appears that WS-Policy offers more possibilities, it could be selected to fit the OWL-S framework. This is the usual standard battle.

<sup>16</sup> WSLA: Web Service Level Agreement

<sup>17</sup> Web Services Agreement specification from the Open Grid Forum (OGF), March 2007

**Benefits:** Benefits can be drawn from the various usage scenarios of this technology. Scenario 1, service selection: In B2B or B2C environment, different offers can provide similar service functionality. Customers require tools to decide which of those services fits best their needs in terms of functional and non-functional characteristics. As a result, the customer takes informed decisions. For instance, in the Lunch Food service example previously mentioned, the customer may prefer, for a slightly higher price to have the food delivered and maintained warm without having to provide local ovens. The fact to use semantics allows to draw conclusions based on imprecise data (ex: slightly higher price). Scenario 2<sup>18</sup>, QoS monitoring: At runtime, measurement is used to detect the real behavior of the service. If the service does not perform as expected, real time actions can be taken to improve it. This way, the service provider reacts in real time to fulfill service's requirements. This is the case, for instance, when the response time of a service is higher than expected. In that case the system should detect it and take appropriate action (for instance add another cpu in the cluster) to conform to the agreed SLA. Scenario 3, QoS parameters sizing: Today, it is difficult for a service provider to set the QoS parameters of his services, most of the time they are set in an empiric way. By consolidating the QoS parameters of the various services involved in the delivery of a service, the service provider will provide more precised values for the service parameters. This is the case for instance when a company is offering a business service through the web, which is implemented using ten software services, distribute on three internal clusters. Those clusters are also supporting other applications. The final response time depends on many parameters. If each service has information about its response time (ex: a range from minimum to maximum response time), the consolidation of this information at the highest service level would provide information that is more accurate.

## **Conclusions**

The evolution of infrastructure middleware is tightly link to the evolution of the Service entity. After a deep look at the service model, it appears that major technical trends (SCA, semantics) have a direct impact on the descriptive part of the service model and, as a consequence, on the middleware infrastructure. A richer service model reduces development effort, improves service interoperability and discovery, and allows service composition and better management of QoS.

## **Consequences on the service model**

It is interesting to consolidate into one drawing the impact of the various technologies that were considered in this paper on the service model. The resulting service model is represented in Figure 9. It is worth comparing it to the original model represented in Figure 1. It is easy to see that the original WSDL file is now complemented by much new functionality provided by UDEF keys technology, SCA configuration file, and the OWL-S semantic language. It should be noted that it is the descriptive part of the service model that has evolved, which is precisely the part that is handled by tools and middleware infrastructure.

---

<sup>18</sup> it is important to know the relationship between services in order to consolidate the SLAs for the service that depends on one or several.

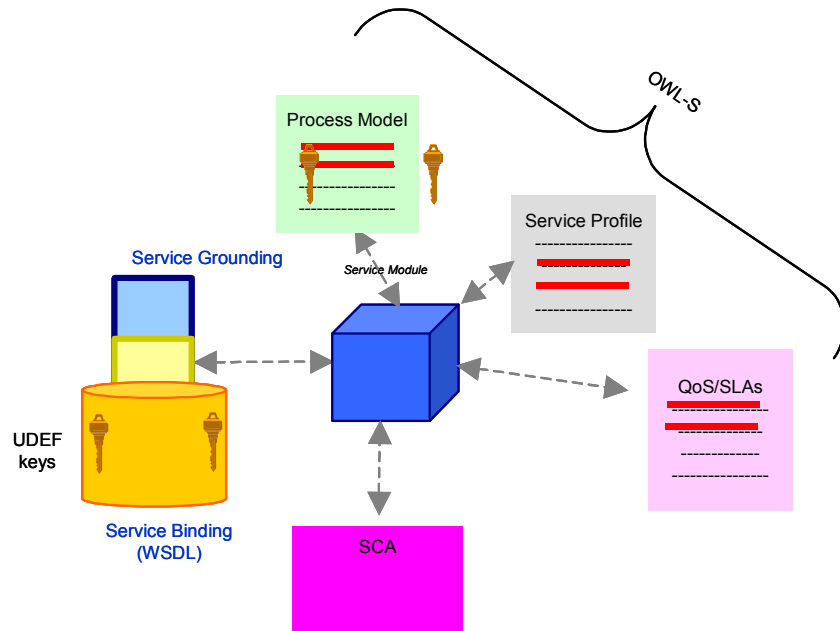


Figure 9: The resulting service model

To provide a global and more detailed vision of the functions provided by the descriptive parts, everything has been consolidated into a set of logical files as represented in Figure 10. It should be noted that all this information may be implemented in one file, in several files, or it may appear as metadata in a service repository.

**WSDL file** Syntactic description – *Service Binding*  
List of Services  
Parameter lists (input/output)  
Mapping to/from message format

**UDEF keys** Data Semantic – *Service interoperability*  
Input/Output parameters  
are UDEF-indexed

**Service configuration** SCA – *Building composite services*  
References  
Properties  
Composites

**Service Grounding** OWL-S – *Service binding*  
Ontology of data types  
Process Model

**Service Semantic** Semantic description - *Service discovery / Service composition*  
<serviceDescription>  
.....  
<preCondition>  
<result>

**Service Attributes** QoS *Service selection / Service sizing*  
<serviceCategory>  
<serviceParameters>  
<serviceClassification>  
<serviceProduct>

Figure 10: Comprehensive Service Description

## **Some general comments**

The introduction of ontology means strong formalism being applied to data. History has shown that many attempts done by academia have failed to catch on, mostly because of the complexity of formal reasoning models and of formal definitions for even simple descriptions. Then, one can legitimately wonder if OWL-S will succeed. This is an everlasting debate. The world is complex and to model it, complex models are required. In order to apply intelligence, reasoning is expressed through rules that execute against some data. This data needs a certain level of formalism. Which one is the most appropriate will be decided by experience. The goal of Ontology is to try to represent the world. It starts from the concept of data and wants to express relationships between this data. Simple hierarchical relationship is too limited (ex: XML) so network type of relationship was introduced (ex: RDF). However, this appeared to be not enough. Relationships exist but under certain conditions, thus those conditions are expressed through rules. Rules bring dynamicity and allow to reason about things. The key objective of ontology is the ability to reason.

Is it possible to avoid the use of semantic by specifying industry specific interfaces that can be agreed in industry bodies? Obviously not. Many standards exist but the programmers still spend time on understanding the real meaning of the data. UDEF is a way to start addressing this problem but it was shown that even with well defined interfaces (using UDEF keys) we will not automate the mapping between an Employee entity and a Traveler entity without intelligence and an ontology that will express that a Traveler can also be an Employee. The trick here is to progress step by step, not trying to solve the entire world problem at once.

Formalisms and standards are good and bad. One of the bad aspects is the fact that they may block innovation. This is quite often the case when the formalism or standard does not support evolution. One advantage of OWL-S is that it is a framework in which other technologies or standards can fit in. Other models could replace the recommended ontology without affecting the overall framework. This ensures an evolution path.

Another aspect is the notion of performance. The middleware becoming "thicker" this raises the fear of a slower execution of the applications sitting on it. At this stage it is interesting to note that each time a new set of technologies is introduced, the issue of performance is raised. Each new release of Windows requires more CPU and memory but PCs cope with that. The mere fact of adding more functionality to software implies more CPU cycles. However, it is possible to challenge the assertion that SOA is potentially slow today. BPEL is a key component of SOA and the fear of customers is that it will run very slowly. When I participated to the first very large SOA project in EMEA at Turkcell, many measures were done to check this aspect. In fact, it appeared that the BPEL PM was able to cope with all the incoming requests at the surprise of everybody, including me.

## **Consequences on SOA middleware business**

*For IDC, the SOA software market will grow at a speed of 48% for the next 3 years to reach \$14,2B by 2011*

An IDC report<sup>19</sup> published in May 2007, forecasted the Worldwide SOA-driven software market to grow from \$2B in 2006 to \$4,4B in 2008, and to \$14,2B by 2011, representing a CAGR<sup>20</sup> of 48%.

---

<sup>19</sup> WorldWide SOA-Driven Software 2007-2011 Forecast – IDC May 2007

<sup>20</sup> CAGR: Compound Aggregated Growth

AMR<sup>21</sup> observed that SOA spending is significant — In 2007 alone, "the average SOA adopter spent nearly \$1.4M, and 45% reported spending over \$500K on SOA software and services," the consultancy observed. By 2012, 77% of companies will have SOA, the analyst firm predicts. The only things that could diminish SOA adoption are a bad economy that results in IT projects getting cut, or a shortage of skills for building, deploying, and maintaining SOA-based applications, the analyst firm adds.

It is interesting to note that Analysts' forecasts are based on existing products. They do not take into account new technologies that will appear in the next coming years. In any case, only SCA technology will be fully available three years from now. This is unlikely to be the case for semantics technologies. Thus, data about those new technologies do not exist. However, those technologies will develop in a larger and mature SOA markets, since by that time 77% of companies will be using the service approach. For Yefim Natis<sup>22</sup>, "In a mature SOA environment, the new imperative will focus on the inner intelligence of the SOA Services". Thus, semantics is announced. To refine Y. Natis's statement, we suggest that services are unlikely to perform reasoning themselves. They will carry the knowledge under the form of ontology and rules and communicate it to a middleware component to perform the reasoning. For instance, a business service may ask middleware to identify a service having a given set of characteristics. In this case, the requester service will communicate to the middleware rules engine the set of characteristics and will expect to get an answer from it.

Hence, what companies do expect from getting into the SOA bandwagon? Their expectations were and still are:

- Greater business agility
- Reuse of existing assets
- Lower IT cost.

It should be expected that any technology that can significantly contribute to one or several of those three objectives would be successful. It has been explained in this paper how those various goals can be addressed with appropriate technologies. Business agility means the ability to adapt existing business processes to new needs. We have shown that SCA provides service dependencies map that allows evaluating the impact of change in the service architecture. We saw also that a better service description, thanks to UDEF and OWL-S, would allow achieving dynamic service identification and invocation. This last key feature allows adding new services to a running environment without stopping it. Thus, SCA, UDEF and OWL-S will significantly contribute to improve agility.

For businesses, "*ROI*<sup>23</sup> has become the most important criteria for SOA going forward. However, measuring return of investment for a SOA project is the greatest challenge facing developers working on SOA programs.... Savings from services reuse is the most common benefit, but the impact on business agility or improved business processes is tough to measure".

Service reuse is very low today but is still a key SOA objective. We presented technologies that would greatly contribute to improve the current situation: SCA will

---

<sup>21</sup> See Full PDF report available at [http://www.soainaction.com/blog/2008/07/soa\\_market\\_will\\_double\\_in\\_four.php](http://www.soainaction.com/blog/2008/07/soa_market_will_double_in_four.php)

<sup>22</sup> "Context Delivery Architecture: Putting SOA in Context" Gartner report published in 2008

<sup>23</sup> Joe McKendrick, ebizQ article, Nov 07, 2008

provide easier reuse of composite services, and UDEF and OWL-S will provide better description of the services to facilitate their discovery and use. Reuse appears in two different contexts: internal to an enterprise and on the web. Up to now, reusability has been considered mainly inside the enterprise and all the statistics refer to that case. In the coming years, reusability will have to be considered at the Internet level because we shall be talking about applications built (statically or dynamically) by combining services taken from multiple web sites and developed by other companies. This type of reusability will only be possible if technologies such as UDEF and OWL-S are well spread. Thus, reusability is also a long-term objective for SOA.

Technologies are key to achieve agility and reuse but they would be useless if governance is not put in place. Working methods and good organization are mandatory. For Gartner<sup>24</sup>, *“the focus should be on creating shared services and the governance processes for sharing ... Large organizations are challenged to create enterprise governance”*.

A last point worth mentioning is the lack of SOA expertise (architects). For Gartner<sup>24</sup> *“Even if a valid business case exists, then the required skills are often unavailable in-house, and the costs and effort to develop in-house skills and acquire outside expertise are often daunting”*.

## General conclusion

Semantic is getting more and more momentum in the IT world today. Many solutions cannot be automated because of a lack of understanding of the context and of the components involved. Semantic is about giving data about data (i.e. information), data about services, and allowing reasoning on this meta-data. To achieve that, metadata needs to be structured into what is called an ontology that provides a representation of the world and, an associated language to process it needs to be defined. Semantics provides reasoning capabilities to IT system. It was shown in this paper that the metadata belongs to the descriptive part of the service model and that it requires new components in the middleware infrastructure to interpret it. Hence, the advent of semantics translates into an expansion of the middleware infrastructure layer. It is hard to predict when the semantic wave will take place, Figure 11 proposes a timeline, but it should be a major wave.

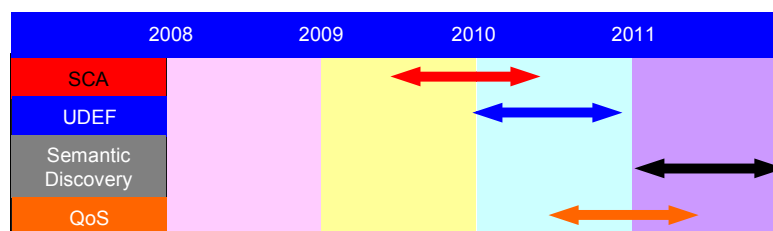


Figure 11: A possible timeline for the introduction of new technologies in the IT business World

<sup>24</sup> Gartner, Christy Pettey, November 3, 2008, <http://www.gartner.com/it/page.jsp?id=790717>