Service Oriented Architectures for convergent Service Delivery Platforms

Service Oriented Architecture and Telcos

Authors:

<table>
<thead>
<tr>
<th>Name</th>
<th>Organization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ferenc Telbisz, Balázs Gódor</td>
<td>Magyar Telekom Telecommunications Plc.</td>
</tr>
<tr>
<td>Alan Ryan</td>
<td>eircom</td>
</tr>
<tr>
<td>Sophie Cherki, Pierre-Arnaud Muller</td>
<td>France Télécom</td>
</tr>
<tr>
<td>Sune Jakobsson</td>
<td>Telenor ASA</td>
</tr>
</tbody>
</table>

Abstract

One of the recently most talked about topics within the Telecom/IT industry community is the subject of Service Delivery Platforms (SDPs). These platforms aim to enable Telecom operators to provide a complete environment for creation, deployment, execution, management and billing for a wide range of value added content and services. Services deployed using these platforms will be agnostic to the underlying network technology. This requires a new, more flexible, architecture and a common communication framework that will support interoperability. A promising approach to this is to use a Service Oriented Architecture (SOA). SoA is a software architecture involving loosely coupled, location independent services generally using the so-called "find-bind-execute" paradigm for the communication between SOA service providers, SOA service users and a SOA service registry. An essential characteristic of an SOA is that it provides published contract-based, platform and technology neutral service interfaces. This means that the interface of a service is independent of its implementation. In this paper the principles of the Service-Oriented Architecture are described, as well as the relationship between the Enterprises and SOA. The potential benefits of SOA, its relevance and its limitations to a Telco are also discussed. The document is most useful for CIO-s, IT technology and business management specialists.
Eurescom participants in project P1652 are:

- Portugal Telecom Inovação
- Magyar Telekom Telecommunications Plc.
- Telenor ASA
- NTT Japan
- *eircom*
- France Télécom Group

[Project title] Service Oriented Architecture for convergent Service Delivery Platforms

[Document title] Service Oriented Architecture and Telcos

Editor: Alan Ryan, Eircom

Project leader: Nuno Silva, Portugal Telecom

Project supervisor: Anastasius Gavras

Eurescom published project result; EDIN 0531-1652

© 2006 Eurescom participants in project P1652

Disclaimer

This document contains material, which is copyright of certain Eurescom PARTICIPANTS and may not be reproduced or copied without permission. The information contained in this document is the proprietary confidential information of certain Eurescom PARTICIPANTS and may not be disclosed except in accordance with Section 5 of the Eurescom’s general conditions of the contract.

The commercial use of any information in this document may require a licence from the proprietor of that information. The material of the document can be used for specific purposes, namely to support the work of the TC5 committee in 3GPP/ETSI.

Neither the PARTICIPANTS nor Eurescom warrant that the information contained in this document is capable of use, or that use of the information is free from risk, and accept no liability for loss or damage suffered by any person using the information.
Preface

Along with the IP Multimedia Subsystem (IMS), one of the most recently talked about topics within the Telecom/IT industry community is the subject of Service Delivery Platforms (SDPs). These platforms aim to enable telecom operators providing a complete environment for creation, deployment, execution, management and billing for a wide range of value added content and services. Services deployed using these platforms will be agnostic to the underlying network technology. In addition, SDPs aim to open up the operators’ network in a secure robust manner such that 3rd party application developers can provide a potentially vast array of content and services to the operators subscribers.

Many of the capabilities needed to build these new services are scattered among different Service Delivery Platforms (SDPs): real-time charging, location, presence, buddy list management, streaming, etc. The challenge we face is how to expose these capabilities of the different platforms, and access them in a uniform way to compose new services. From the perspective of a developer (internal or 3rd party), the problem can be viewed as the need to integrate the different architectures that govern enterprise and communication environments, an important consideration, given the convergence of content and the traditional telecom environment.

This requires a new, more flexible, architecture and a common communication framework that will support interoperability between these two diverse worlds. A leading approach is to use a Service Oriented Architecture (SoA), with XML/WSDL as the underlying communication framework. SoA is a software architecture involving loosely coupled, location independent services generally using the so-called "find-bind-execute" paradigm for the communication between SoA service providers, SoA service users and a SoA service registry. Any given service may assume a client or a server role with respect to another service, depending on situation. An essential characteristic of an SoA is that it provides published contract-based, platform and technology neutral service interfaces. This means that the interface of a service is independent of its implementation. In practice, interfaces are defined using ubiquitous IT standards such as XML, HTTP, SOAP, and WDSL. Major goals of an SoA in comparison with other software architectures used in the past are to enable faster adaptation of software to changing business needs, cost reduction in the integration of new services, as well as in the maintenance of existing services.

From this background this study aims at reviewing the important enabling role that Web Services technologies play in an SoA implementation, and examine how an SoA can change the way new services are created, both from the technical and business perspectives. Furthermore the study examines how service orchestration enables operators to easily bring together components in SDPs and external systems intelligently to deliver services to the users and finally to identify the challenging issues and impacts of introducing an SoA at the Service Layer;

The study P1652 “Service Oriented Architectures (SoA) for convergent Service Delivery Platforms (SDPs)” started in June 2006 and concluded in December 2006. The study was executed with an overall resources of 14 person-months and the participants were Portugal Telecom Inovação, SA, France Télécom, eircom Ltd., Telenor R&D, Magyar Telekom Telecommunications Ltd. and NTT Japan. Mr. Nuno Silva from Portugal Telecom Inovação, SA, was leading the study.

This document is the first of two study deliverables.

<table>
<thead>
<tr>
<th>No</th>
<th>Deliverable Title</th>
<th>Issue date</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1 – EDIN 0531-1652</td>
<td>Service Oriented Architectures and Telcos</td>
<td>December 2006</td>
</tr>
<tr>
<td>D2 – EDIN 0532-1652</td>
<td>Applying Service Oriented Architectures to Service Delivery Platforms</td>
<td>December 2006</td>
</tr>
</tbody>
</table>
Executive Summary

It is expected over the next few years that most Telecoms Operators will transform their existing portfolio of applications (OSS and BSS) to meet the emerging requirements in the marketplace. This transformation is being driven by the continued up take of xDSL, FTTx and wireless services, the fall-off in revenues and profitability from traditional voice and data services, and the corresponding technology transformation in the core network.

The route to transformation has many paths and each operator will choose its own route based upon its particular needs and circumstances. Most will probably adopt an evolutionary approach based upon an internal consolidation of existing systems to ease integration of new applications and improve operational efficiency. To further reduce costs and speed up delivery of services operators will also need to substantially increase their exploitation of standardised components and reuse existing and proven software capabilities.

Systems integration continues to remain a key skill throughout this evolution. OSS and SDP (Service Delivery Platform) vendors, in particular, will need to provide strong capabilities in consulting, implementation and support services.

The market today for Service Delivery architectures is characterised by vast portfolio of diverse components. To achieve cost effective interoperability, open interfaces are required and therefore compliance to recognised industry standards will be essential. The rising adoption of SDP and IMS solutions will contribute to an industry wide standardisation of the Service Delivery architecture.

Telecoms Operators have not been standing still, most already use, or have started to use IT architecture concepts based upon components and according to a Service Oriented Architecture (SOA). SOA is primarily intended to provide business-level software modularity and rapid reuse of software components. SOA describes modular and distributed software that offers well-described interfaces. SOA differs from the traditional client-server model in its emphasis on loose coupling and its use of separately standing interfaces. Web Services is considered to be a standardised subset of an SOA. Operators expect an SOA to deliver agility and a faster time to market for new services and features, but only once a basic set of re-useable components are already available.

The maturity of adoption varies among and within Telecoms Operators with large Operators having had some kind of SOA in place for several years, whilst small to mid-size Operators are at the early stages of introduction. Most are using Web Services as the key enabling technology and are deploying simple services behind the firewall in order to familiarise themselves with this new technology. Many new operators have had the privilege of starting from scratch, inventing their architecture with a new and young team, an SOA with Web Services tends to be their preferred approach.

Some key technical challenges still remain, such as, the standardisation of data models and protocols within the enterprise to a single information model. The availability of a reliable/credible information model is a critical success factor in the realisation of the full benefits of an SOA. Other challenges exist in modelling & design methodologies and service deployment & evolution. It is expected that these will be overcome in the foreseeable future and should not be viewed as a reason to delay adoption of either SOA or Web Services technologies. It must also be noted that SOA is not a ‘one model fits all’ solution. Processes that consume only limited amounts of data and with response times typically in the low seconds are best suited to an SOA whilst those processes that handle massive volumes of data or have real-time response characteristics are best left to more traditional approaches.

To summarise, SOA is based upon tried and proven software engineering principles developed (though not necessarily implemented) over the last 30 years. Web Services provides a standards-based cross-platform interoperability suite that has significant industry momentum. Operators and ISVs are embracing both technologies which are expected to become mainstream in the next few years. Some challenges still exist though they are not insurmountable.
Table of contents

Preface .................................................................................................................................................. 3
Executive Summary................................................................................................................................. 4
Table of contents................................................................................................................................... 5
List of Figures.......................................................................................................................................... 7
Abbreviations......................................................................................................................................... 8
1 Introduction.......................................................................................................................................... 9
2 Principles of Service-Orientation ....................................................................................................... 11
   2.1 Reusable Services ....................................................................................................................... 11
   2.2 Formal Contracts Among Services ........................................................................................... 12
   2.3 Loosely coupled Services ........................................................................................................... 13
   2.4 Service abstraction ..................................................................................................................... 13
   2.5 Service Autonomy ..................................................................................................................... 14
   2.6 Service composability ............................................................................................................... 15
   2.7 Stateless Services ..................................................................................................................... 16
   2.8 Service Discovery .................................................................................................................... 16
   2.9 Interrelation of the principles ................................................................................................... 16
3 Underlying technologies: the Web Services .................................................................................... 17
   3.1 The model of Web Services ........................................................................................................ 17
   3.2 Roles and Operations in a Web Services Architecture ............................................................... 18
   3.3 Web Services Technologies ....................................................................................................... 19
       3.3.1 XML (Extensible Markup Language) .................................................................................. 19
       3.3.2 HTTP (Hypertext Transfer Protocol) ................................................................................. 19
       3.3.3 SOAP (Simple Object Access Protocol) ............................................................................ 19
       3.3.4 WSDL (Web Services Description Language) ................................................................. 20
       3.3.5 UDDI (Universal Description, Discovery and Integration) ............................................ 20
       3.3.6 WSFL (Web Services Flow Language) and BPEL ......................................................... 20
       3.3.7 REST (Representational state transfer) ............................................................................. 21
   3.4 The Web Services Architecture .................................................................................................. 21
   3.5 Layer responsibilities in the Web Services architecture ........................................................... 23
       3.5.1 Network Layer ..................................................................................................................... 23
       3.5.2 XML based messaging layer .............................................................................................. 23
       3.5.3 Service description layer ................................................................................................... 23
       3.5.4 Service publication layer ................................................................................................... 23
       3.5.5 Service Discovery Layer .................................................................................................... 23
       3.5.6 Service Flow layer ............................................................................................................. 24
       3.5.7 Vertical Function Sets ......................................................................................................... 24
4 Web Services and Enterprises ............................................................................................................ 25
   4.1 Current State of the Art: ............................................................................................................. 26
       4.1.1 Technical challenges remain .............................................................................................. 26
   4.2 Where is the Industry leading us ? ............................................................................................ 26
       4.2.1 Web 2.0 ............................................................................................................................. 26
       4.2.2 Software as a Service (SaaS): .......................................................................................... 27
       4.2.3 Event Driven Architectures (EDA) .................................................................................. 28
   4.3 Emerging business requirements ................................................................................................ 29
       4.3.1 Drivers: ............................................................................................................................... 30
       4.3.2 Responses and implications: ............................................................................................ 30
       4.3.3 Challenges and Critical Success Factors: ........................................................................ 31
       4.3.4 Case of the legacy applications ....................................................................................... 32
       4.3.5 Business objectives .......................................................................................................... 32
       4.3.6 Overall benefits of Service-Oriented Architecture .......................................................... 33
4.4 Common Information Model in SOA ............................................................................................. 33
   4.4.1 What is a single information model? ..................................................................................... 33
   4.4.2 Single information model and SOA? .................................................................................... 34
   4.4.3 Example ............................................................................................................................... 34
   4.4.4 Comments ........................................................................................................................... 34
   4.4.5 Conclusion ............................................................................................................................ 34
5 Relevance and limitations to a Telco ............................................................................................... 35
   5.1 SOA relevance to the business and operations of a Telco ......................................................... 35
       5.1.1 A new step in software engineering progress............................................................... 35
© 2006 Eurescom participants in project P1652
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1.2</td>
<td>Some infrastructure products are available to implement an SOA</td>
<td>36</td>
</tr>
<tr>
<td>5.1.3</td>
<td>Impacts on the enterprise</td>
<td>36</td>
</tr>
<tr>
<td>5.1.4</td>
<td>The relevance of SOA for OSS/BSS must be supported by adequate governance</td>
<td>37</td>
</tr>
<tr>
<td>5.2</td>
<td>SOA experiment in Telco</td>
<td>38</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Analysis of SOA applied to a Telco</td>
<td>38</td>
</tr>
<tr>
<td>5.2.2</td>
<td>SOA to support the exchanges between information system and service platforms</td>
<td>38</td>
</tr>
<tr>
<td>5.2.3</td>
<td>VNO</td>
<td>39</td>
</tr>
<tr>
<td>5.2.4</td>
<td>IPTV</td>
<td>39</td>
</tr>
<tr>
<td>6</td>
<td>Conclusions and Recommendations</td>
<td>41</td>
</tr>
<tr>
<td>References</td>
<td></td>
<td>42</td>
</tr>
</tbody>
</table>
List of Figures

Figure 1 Gartner Hype curve for 2005 ................................................................. 10
Figure 2 A reusable service exposes reusable operations .............................. 12
Figure 3 Service contracts formally define the service, operation, and message components of a service-oriented architecture ................................................................. 12
Figure 4 Services limit dependencies to the service contract, allowing underlying provider and requestor logic to remain loosely coupled ................................................. 13
Figure 5 Service operations abstract the underlying details of the functionality they expose .... 14
Figure 6 Autonomous services have control over underlying resources .............. 15
Figure 7 UpdateEverything operation showing a service composition ............. 16
Figure 8 Web Services Model ............................................................................. 18
Figure 9 Layered architecture of Web Services ................................................. 22
Figure 10 Positioning SOA services with regards to objects, components & process 35
Figure 11 The closed loop enterprise ................................................................. 37
Figure 13 Network architecture of caller ID service on IPTV .......................... 40
**Abbreviations**

CORBA  Common Object Request Broker Architecture  
FTP     File Transfer Protocol  
HTTP    Hypertext Transfer Protocol  
IETF    Internet Engineering Task Force  
IT      Information Technology  
IT      Information Technology  
JSON    JSONJavaScript Object Notation  
MQSeries An IBM software family (middleware)  
OASIS   Organization for the Advancement of Structured Information Standards  
REST    Representational state transfer  
RPC     Remote Procedure Call  
RSS     RDF Site Summary  
SGML    Standard Generalized Markup Language  
SMTP    Simple Mail Transfer Protocol  
SOA     Service Oriented Architecture  
SOAP    Simple Object Access Protocol  
UDDI    Universal Description, Discovery and Integration  
W3C     World Wide Web Consortium  
WSDL    Web Services Description Language  
WSFL    Web Services Flow Language  
XHTML   XML (version of) HTML  
XML     Extensible Markup Language  
XSD     Extensible Schema Definition (Language)  
YAML    Yet Another Markup Language  
VoIP    Voice over IP  
GTDD    Global Telecommunications Data Dictionary  
ITU     International Telecommunication Union  
OAM&P   Operations, Administration, Maintenance and Provisioning  
ATIS    Alliance for Telecommunications Industry Solutions  
TMF     Telecommunication Management Forum  
ITU-T   The International Telecommunications Union – Telecommunications Sector  
OSS/BSS Operations And Business Support Systems  
MVNO    Mobile Virtual Network Operator  
VNO     Virtual Network Operator  
IPTV    Internet Protocol Television
1 Introduction

The term “service oriented” means that logic required to solve a large problem can be better constructed, carried out and managed if it is decomposed into a collection of smaller, related pieces. Each of these pieces addresses a concern or a specific part of the problem. Service-oriented architecture (SOA) encourages individual units of logic to exist autonomously yet not isolated from each other. Units of logic are still required to conform to a set of principles that allow them to evolve independently while still maintaining a sufficient amount of commonality and standardisation. Within SOA these units are known as services. [1]

Although SOA is heralded as a brand new architecture, in reality it is reinventing and applying well established principles and practices – like structured programming, top-down planning, acknowledgement of messages, the proper use of exception logic, visibility in programs, etc. – already accepted and taught in Computer Science courses at Universities up to 30 years ago, but neglected and thus rarely applied in practice. At the same time SOA often appears as a collection of marketing slogans and has joined the latest crop of industry buzzwords. The emergence of SOA is acknowledging that after decades of extensive use Information Technology, a significant body of application systems have been written to a monolithic model with poor quality attributes such as, adaptability, extensibility, interoperability and scalability. SOA is, however, a much needed ‘new’ way of thinking, designing and working.

A service oriented system requires an implementation platform. Leading corporations with a record of adopting and exploiting new IT paradigms and technologies have shown the way by developing complex service oriented applications as far back as the mid/ late 1990s. One such example is Credit Swiss First Boston’s extensive use of CORBA. Contemporary exploitations of this approach use many web tools developed since the inception of WWW, like XML, SOAP (Simple Object Access Protocol), WSDL (Web Services Description Language), REST (Representational State Transfer), UDDI (Universal Description, Discovery and Integration), etc. SOA can be implemented on other platforms, such as traditional Client/Server; however, the web services platform offers the most competent environment today and offers the best prospect for success.

SOA is a new, and still evolving technology. According to the estimations of the Gartner Group in 2005 [2], it was near the end of the phase of disillusionment (Figure 1). Despite the disillusionment with SOA in 2005, Gartner expected support for an SOA to grow and to mature as a technology within ten years although many changes in user and vendor organisations and technologies are required before SOA reaches its full potential.

This document opens with a short executive summary followed by a brief introduction. Chapter 3 reviews the principles of SOA. Chapter 4 provides a dissertation on the underlying technologies, Web-Services, and the Web Services architectural model necessary for their successful implementation. Chapter 5 discusses the relevance of web services with respect to the Enterprises, the current status of adoption, some key outstanding challenges and requirements, followed by the importance to SOA of a common information model. Chapter 6 introduces the relevance and limitations of SOA for the Telecoms Operator. Finally, in Chapter 7, we present conclusions and recommendations section to complete the document.
Figure 1 Gartner Hype curve for 2005

Source: Gartner (August 2005), © 2005 Gartner, Inc. and/or its Affiliates. All Rights Reserved.
2 Principles of Service-Orientation

The term Service-Oriented Architecture (SOA) expresses a software architecture that defines the use of services to support the requirements of software users. (See. [1]) In a SOA environment, nodes on a network make resources available to other participants in the network as independent services that the participants access in a standardized way. Services are self-contained, reusable software modules with well-defined interfaces and are independent of applications and the computing platforms on which they run.

Most definitions of SOA identify the use of Web services (see Ch. 3), e.g. using SOAP (Ch 3.3.3) or REST (Ch. 3.3.7) in its implementation. However, one can implement SOA using any service-based technology. The OASIS SOA Reference Model Technical Committee [5] is working on defining SOA independent of any specific technologies. The Committee is developing a Reference Model for Service Oriented Architecture [6]. This is due primarily to the fact that SOA is being used as a term in an increasing number of contexts and specific technology implementations. Sometimes, the term is used with differing – or worse, conflicting – understandings of implicit terminology and components. This Reference Model is being developed to encourage the continued growth of different and specialized SOA implementations whilst preserving a common layer of understanding about what SOA is.

SOA helps users build composite applications, which are applications that use the functionality from multiple sources within and beyond the enterprise to support horizontal business processes. It is a style of enterprise IT architecture that enables the creation of applications that are built by combining loosely coupled and interoperable services. These services interoperate based on a formal definition (or contract) that is independent from the underlying run-time platform and programming language. The interface definition hides the language-specific service implementation. A SOA is independent of development technology (such as Java, .NET etc) and is therefore vendor independent.

Although there is no official or defacto set of service-orientation principles, but there are, however, a common set of principles mostly associated with service orientation. Different organizations have published their own versions of service-oriented principles. As a result of this, many variations exist. The most common principles relate to loose coupling, autonomy, discoverability, composable, reuse, service contracts, abstraction, and statelessness. These are reviewed in the following subsections.

2.1 Reusable Services

Services are reusable. Logic is divided into services with the intention of promoting reuse. Service-orientation encourages reuse in all services, regardless of whether immediate requirements for reuse exist. By applying design standards that make each service potentially reusable, the chances of being able to accommodate future requirements with less development effort are increased. Inherently reusable services also reduce the need for creating wrapper services that expose a generic interface over the top of less reusable or accessible service /function.

This principle facilitates all forms of reuse, including inter-application interoperability, composition, and the creation of utility services. Because a service is simply a collection of related operations, it is therefore the logic encapsulated by the individual operations that must be deemed reusable in order to warrant representation as a reusable service.

In Figure 2 an account updating service is shown as an example of reusability. As a service oriented structure is heavily based on messaging, this also indirectly supports service reusability particularly if this is implemented through the use of SOAP headers. For example, SOAP allows for messages to become increasingly self-reliant by grouping metadata details with message content into a single package (the SOAP envelope). Messages can be equipped with processing instructions and business rules that allow them to dictate to recipient services how they should be processed.
The processing-specific logic implied in a message alleviates the need for a service to contain this logic. More importantly, it imposes a requirement that service operations become less activity-specific — in other words, more generic. The more generic a service’s operations are, the more reusable the service is.

![Figure 2 A reusable service exposes reusable operations](image)

2.2 Formal Contracts Among Services

**Services share a formal contract.** They adhere to a communications agreement, as defined collectively by one or more service description documents. These contracts provide a formal definition of:

- the service endpoint
- each service operation
- every input and output message supported by each operation
- rules and characteristics of the service and its operations

Service contracts therefore define almost all of the primary parts of an SOA. Good service contracts may also provide semantic information that explains how a service may go about accomplishing a particular task (Figure 3). Either way, this information establishes the agreement made by a service provider and its service requestors.

![Figure 3 Service contracts formally define the service, operation, and message components of a service-oriented architecture](image)
Because this contract is shared amongst services, its design is extremely important. Service requestors that agree to this contract can become dependent on its definition. Therefore, contracts need to be carefully maintained and versioned after their initial release.

Within the Web services framework, service description documents (such as the WSDL definition, XSD schemas, and policies) can be viewed collectively as a communications contract that expresses exactly how a service can be programmatically accessed.

### 2.3 Loosely coupled Services

**Services are loosely coupled.** They maintain a relationship that minimizes dependencies and only requires that they maintain an awareness of each other.

No one can predict how an IT environment will evolve. How automation solutions grow, integrate, or are replaced over time can never be accurately planned, because the requirements that drive these changes are almost always external to the IT environment. Being able to ultimately respond to unforeseen changes in an efficient manner is a key goal of applying service-orientation. Realising this form of agility is directly supported by establishing a loosely coupled relationship between services.

![Loosely coupled Services](image)

**Figure 4 Services limit dependencies to the service contract, allowing underlying provider and requestor logic to remain loosely coupled**

Loose coupling is a condition wherein a service acquires knowledge of another service while still remaining independent of that service (Figure 4). Loose coupling is achieved through the use of service contracts (service descriptions) that allow services to interact within predefined parameters.

It is interesting to note that within a loosely coupled architecture, service contracts actually tightly couple operations to services. Once a service is formally described as being the location of an operation, other services will depend on that operation-to-service association.

### 2.4 Service abstraction

**Services represent an abstraction of the underlying logic** beyond what is described in the service contract, services hide logic from the outside world. Also referred to as service interface level abstraction, it is this principle that allows services to act as black boxes, hiding their details from the outside world. The scope of logic represented by a service significantly influences the design of its operations and its position within a process.
Figure 5 Service operations abstract the underlying details of the functionality they expose

There is no limit to the amount of logic a service can represent. A service may be designed to perform a simple task, or it may be positioned as a gateway to an entire automation solution. There is also no restriction as to the source of application logic a service can draw upon. For example, a single service can, technically, expose application logic from two different systems (Figure 5).

Operation granularity is therefore a primary design consideration that is directly related to the range and nature of functionality being exposed by the service. Again, it is the individual operations that collectively abstract the underlying logic. Services simply act as containers for these operations.

Service interface level abstraction is one of the inherent qualities provided by Web services. The loosely coupled communications structure requires that the only piece of knowledge services need to interact is each others’ service descriptions.

2.5 Service Autonomy

Services have control over the logic they encapsulate. Autonomy requires that the range of logic exposed by a service exist within an explicit boundary. This allows the service to execute self-governance of all its processing. It also eliminates dependencies on other services, which frees a service from ties that could inhibit its deployment and evolution Figure 6. Upon execution the service has governance over the underlying application logic. This is closely related – practically identical – to the principle of the interface level abstraction of the underlying service logic. Service autonomy is a primary consideration when deciding how application logic should be divided up into services and which operations should be grouped together within a service context.
Autonomy does not necessarily grant a service exclusive ownership of the logic it encapsulates. It only guarantees that at the time of execution, the service has control over whatever logic it represents. We therefore can make a distinction between two types of autonomy.

- **Service-level autonomy.** Service boundaries are distinct from each other, but the service may share underlying resources. For example, a wrapper service that encapsulates a legacy environment that also is used independently from the service has service-level autonomy. It governs the legacy system but also shares resources with other legacy clients.

- **Pure autonomy.** The underlying logic is under complete control and ownership of the service. This is typically the case when the underlying logic is built from the ground up in support of the service.

### 2.6 Service composability

**Services are composable,** i.e. collections of services can be coordinated and assembled to form composite services. A service can represent any range of logic from various types of sources, including other services. The main reason to implement this principle is to ensure that services are designed so that they can participate as effective members of other service compositions, when required. This requirement is irrespective of whether the service itself acts as the composer of others.

A common SOA extension that underlines composability is the concept of orchestration. An update process is shown in Figure 7, which is composed of processes updating the account, its logs and history. Here, a service-oriented process (which essentially can be classified as a service composition) is a parent process service (UpdateEverything) that composes process participants.
The requirement for any service to be composable also places an emphasis on the design of service operations. *Composability is simply another form of reuse* and therefore operations need to be designed in a standardized manner and with an appropriate level of granularity in order to maximize composability opportunities.

### 2.7 Stateless Services

*Services are stateless*, i.e. services should minimize the amount of state information they manage and the duration for which they hold it. State information is data specific to a current activity. While a service is processing a message, for example, it is temporarily stateful. If a service is responsible for retaining state for longer periods of time, its ability to remain available to other requestors will be impeded.

Statelessness is a preferred condition for services and one that promotes reusability and scalability. In order for a service to retain as little state as possible, its individual operations need to be designed with stateless processing considerations. A primary quality of SOA that supports statelessness is the use of document-style messages. The more intelligence added to a message, the more independent and self-sufficient it remains.

### 2.8 Service Discovery

Services should be designed to be outwardly descriptive so that they can be found and assessed via available discovery mechanisms. Designing services so that they are naturally discoverable enables an environment whereby service logic becomes accessible to new potential service requestors. This is why service discoverability is tied closely to the following service-orientation principles:

- Service contracts are what service requestors (or those that create them) actually discover and subsequently assess for suitability. Therefore, the extent of a service’s discoverability can typically be associated with the quality or descriptiveness of its service contract.

- Service reusability is what requestors are looking for when searching for services, and it is what makes a service potentially useful once it has been discovered. A service that isn’t reusable would likely never need to be discovered because it would probably have been built for a specific service requestor in the first place.

### 2.9 Interrelation of the principles

The above reviewed basic principles are not independent principles but are heavily interconnected, each one relates to and supports other principles. This close connection was explicitly mentioned above in case of service abstraction and autonomy, but there are other interrelations, too, the reader is referred to the literature [1].

---

*Figure 7 UpdateEverything operation showing a service composition*

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

### 2.7 Stateless Services

*Services are stateless*, i.e. services should minimize the amount of state information they manage and the duration for which they hold it. State information is data specific to a current activity. While a service is processing a message, for example, it is temporarily stateful. If a service is responsible for retaining state for longer periods of time, its ability to remain available to other requestors will be impeded.

Statelessness is a preferred condition for services and one that promotes reusability and scalability. In order for a service to retain as little state as possible, its individual operations need to be designed with stateless processing considerations. A primary quality of SOA that supports statelessness is the use of document-style messages. The more intelligence added to a message, the more independent and self-sufficient it remains.

### 2.8 Service Discovery

Services should be designed to be outwardly descriptive so that they can be found and assessed via available discovery mechanisms. Designing services so that they are naturally discoverable enables an environment whereby service logic becomes accessible to new potential service requestors. This is why service discoverability is tied closely to the following service-orientation principles:

- Service contracts are what service requestors (or those that create them) actually discover and subsequently assess for suitability. Therefore, the extent of a service’s discoverability can typically be associated with the quality or descriptiveness of its service contract.

- Service reusability is what requestors are looking for when searching for services, and it is what makes a service potentially useful once it has been discovered. A service that isn’t reusable would likely never need to be discovered because it would probably have been built for a specific service requestor in the first place.

### 2.9 Interrelation of the principles

The above reviewed basic principles are not independent principles but are heavily interconnected, each one relates to and supports other principles. This close connection was explicitly mentioned above in case of service abstraction and autonomy, but there are other interrelations, too, the reader is referred to the literature [1].
3 Underlying technologies: the Web Services

As it was mentioned earlier, the main – and up to now almost exclusively used – technology for service oriented implementations is the Web Services technology. The term "Web Services" can be confusing as it is, unfortunately, often used in many different ways. Compounding this confusion is the generic term "services" that has a different meaning to the term "Web Services." The term Web Services refers to the technologies that allow for making connections. Services are what one connect together using Web Services. A service is the endpoint of a connection. A more exact definition is the following: enable interoperability (the ability to exchange data through the use of shared data formats and common protocols). ISO/IEC 2382-01 provides a concise definition: "The capability to communicate, execute programs, or transfer data among various functional units in a manner that requires the user to have little or no knowledge of the unique characteristics of those units". Services are what one connects together using interoperability technologies, such as, Web Services. A service is therefore the endpoint of a connection. A more exact definition follows:

A Web service is an interface that describes a collection of operations that are network accessible through standardized XML messaging. A Web service is described using a standard, formal XML notion, called its service description. It covers all the details necessary to interact with the service, including message formats (that detail the operations), transport protocols and location. The interface hides the implementation details of the service, allowing it to be used independently of the hardware or software platform on which it is implemented and also independently of the programming language in which it is written. This allows and encourages Web Services-based applications to be loosely coupled, component-oriented, with cross-technology implementations. Web Services fulfill a specific task or a set of tasks/operations. They can be used alone or with other Web Services to carry out a complex aggregation or a business transaction [8].

Today's applications typically communicate using Remote Procedure Calls (RPC) between objects like DCOM and CORBA, but HTTP was not designed for this. RPC represents a compatibility and security problem; firewalls and proxy servers will normally block this kind of traffic.

A better way to communicate over the internet between applications is by using HTTP, because HTTP is supported by all Internet browsers and servers and is firewall friendly. Web services technologies were created to exploit this key attribute. They provide a way to communicate between applications running on different operating systems, with different technologies and programming languages whilst using the internet as the transport. Indeed the term 'web service' itself seems to be used to describe a number of different types of applications from, at one end of the spectrum, an evolution beyond web interfaces that use URL encoded parameters through to complex, orchestrated business processes. While one doesn't preclude the other, the requirements for each are quite different.

3.1 The model of Web Services

The Web Services architecture is based upon the interactions between three roles: service provider, service registry and service requestor. The interactions involve three distinct operations: (i) publish; (ii) find; and (iii) bind operations. Together, these roles and operations act upon the Web Services artifacts: the Web service software module and its description. In a typical scenario, a service provider hosts a network-accessible software module (an implementation of a Web service). The service provider defines a service description for the Web service and publishes it to a service requestor or service registry. The service requestor uses a find operation to retrieve the service description locally or from the service registry and uses the service description to bind with the service provider and invoke or interact with the Web service implementation. Service provider and service requestor roles are logical constructs and a service can exhibit characteristics of both. Figure 8 illustrates these operations, the components providing them and their interactions [7].
3.2 Roles and Operations in a Web Services Architecture

- **Service provider**: From a business perspective, this is the owner of the service. From an architectural perspective, this is the platform that hosts access to the service, and the application(s) that deliver the service capabilities.

- **Service requestor**: From a business perspective, this is the business that requires certain functions to be satisfied. From an architectural perspective, this is the application that is looking for and invoking or initiating an interaction with a service. The service requestor role can be played by a browser driven by a person or a program without a user interface, for example, another Web service.

- **Service registry**: This is a searchable registry of service descriptions where service providers publish their service descriptions. Service requestors find services and obtain binding information (in the service descriptions) for services during development for static binding or during execution for dynamic binding. For statically bound service requestors, the service registry is an optional role in the architecture, because a service provider can send the description directly to service requestors. Likewise, service requestors can obtain a service description from other sources besides a service registry, such as a local file, FTP site, Web site, Advertisement and Discovery of Services (ADS) or Discovery of Web Services (DISCO).

For an application to take advantage of Web Services, three behaviors must take place: publication of service descriptions, lookup or finding of service descriptions, and binding or invoking of services based on the service description. These behaviors can occur singly or iteratively. In detail, these operations are:

- **Publish**: To be accessible, a service description needs to be published so that the service requestor can find it. Where it is published can vary depending upon the requirements of the application.

- **Find**: In the find operation, the service requestor retrieves a service description directly or queries the service registry for the type of service required. The find operation can be involved in two different lifecycle phases for the service requestor: at design time to retrieve the service’s interface description for program development, and at runtime to retrieve the service’s binding and location description for invocation.

- **Bind**: Eventually, a service needs to be invoked. In the bind operation the service requestor invokes or initiates an interaction with the service at runtime using the binding details in the service description to locate, contact and invoke the service. [8]
3.3  Web Services Technologies

3.3.1  XML (Extensible Markup Language)

The XML is a World Wide Web Consortium’s (W3C) recommended [9] general-purpose, simple, flexible and text format markup language for creating special-purpose markup languages, capable of describing many different kinds of data. In other words, XML is a way of describing data. An XML file can contain the data too, as in a database. It is a simplified subset of Standard Generalized Markup Language (SGML, ISO 8879). Its primary purpose is to facilitate the sharing of data across different systems, particularly systems connected via the Internet. Languages based on XML (for example, Geography Markup Language (GML), RDF/XML, RSS, Atom, MathML, XHTML, SVG, EAD, Klip and MusicXML) are defined in a formal way, allowing programs to modify and validate documents in these languages without prior knowledge of their particular form.

3.3.2  HTTP (Hypertext Transfer Protocol)

Hypertext Transfer Protocol (HTTP) is a method used to transfer or convey information on the World Wide Web. It is a patented open internet protocol whose original purpose was to provide a way to publish and receive HTML pages. Development of HTTP was coordinated by the W3C and working groups of the IETF, culminating in the publication of a series of RFCs, most notably RFC 2616, which defines HTTP/1.1, the version of HTTP in common use today.

HTTP is a request/response protocol between clients and servers. The originating client, such as a web browser, or other end-user tool, is referred to as the user agent. The destination server, which stores or creates resources such as HTML files and images, is called the origin server. In between the user agent and origin server may be several intermediaries, such as, proxies, gateways and tunnels.

An HTTP client initiates a request by establishing a Transmission Control Protocol (TCP) connection to a particular port on a remote host (port 80 by default). An HTTP server listening on that port waits for the client to send a request message.

Upon receiving the request, the server sends back a status line, such as "HTTP/1.1 200 OK", and a message of its own, the body of which is perhaps the requested file, an error message, or some other information.

Resources to be accessed by HTTP are identified using Uniform Resource Identifiers (URIs) (or, more specifically, URLs) using the http: or https: URI schemes.

3.3.3  SOAP (Simple Object Access Protocol)

SOAP is a protocol for exchanging XML-based messages over a computer network, normally using HTTP. SOAP is a lightweight protocol for exchange of information in a decentralized, distributed environment. It is an XML based protocol that consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined data types, and a convention for representing remote procedure calls and responses. SOAP forms the foundation layer of the Web services stack, providing a basic messaging framework that more abstract layers can build on.

There are several different types of messaging patterns in SOAP, but by far the most common is the Remote Procedure Call (RPC) pattern, in which one network node (the client) sends a request message to another node (the server), and the server immediately sends a response message to the client.

Roger Costello has identified two major alternative patterns depending on whether they are action or data oriented. He has defined action-oriented messaging as basically a procedure call, i.e. an action identifier that defined the intent of the message and a number of parameters. Data-oriented messages simply carry data, with the intent of the message being determined independently of the message.
A key benefit of discovering patterns is that they provide a common terminology for communicating issues and trade-offs. The nascent state of web services means that this terminology has still to be refined.

From a network transport perspective both SMTP and HTTP are valid application layer protocols for SOAP, but HTTP has gained wider acceptance as it works well with today's Internet infrastructure; specifically, SOAP works well with network firewalls. This is a major advantage over other distributed protocols like GIOP/IIOP or DCOM which are normally filtered by firewalls. The SOAP specification is currently maintained by the XML Protocol Working Group of the W3C.

3.3.4 WSDL (Web Services Description Language)

WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate, however, the typical bindings with WSDL are SOAP 1.1, HTTP GET/POST, and MIME. [10]

3.3.5 UDDI (Universal Description, Discovery and Integration)

UDDI describes a registry of Web services and programming interfaces for publishing, retrieving, and managing information about services described therein. In fact, UDDI itself is a set of Web services! The UDDI specification defines services that support the description and discovery of (1) businesses, organizations, and other Web services providers, (2) the Web services they make available, and (3) the technical interfaces which may be used to access and manage those services. UDDI is based upon several other established industry standards, including HTTP, XML, XML Schema (XSD), SOAP and WSDL. [11]

3.3.6 WSFL (Web Services Flow Language) and BPEL

IBM and Microsoft had each defined their own, fairly similar, 'programming in the large' languages, WSFL and XLANG, respectively. IBM and Microsoft decided to combine these languages into a new language, BPEL4WS. In April 2003, BEA Systems, IBM, Microsoft, SAP, and Siebel Systems submitted BPEL4WS 1.1 to OASIS for standardization via the Web Services BPEL Technical Committee.

WSFL is an XML language for the description of Web Services compositions. WSFL considers two types of Web Services compositions:

- The first type specifies the appropriate usage pattern of a collection of Web Services, in such a way that the resulting composition describes how to achieve a particular business goal; typically, the result is a description of a business process.

- The second type specifies the interaction pattern of a collection of Web Services; in this case, the result is a description of the overall partner interactions.

However, WSFL has now been superseded by Business Process Execution Language (BPEL). BPEL is a business process language that grew out of WSFL and XLANG. It is serialized in XML and aims to enable “programming in the large”. The concepts of programming in the large and programming in the small distinguish between two aspects of writing the type of long-running asynchronous processes that one typically sees in business processes.

Programming in the large generally refers to the high-level state transition interactions of a process—BPEL refers to this concept as an Abstract Process. A BPEL Abstract Process represents a set of publicly observable behaviors in a standardized fashion. An Abstract Process includes information such as when to wait for messages, when to send messages, when to compensate for failed transactions, etc. Programming in the small, in contrast, deals with short-lived programmatic behavior, often executed as a single transaction and involving access to local logic and resources.
such as files, databases, etc. BPEL’s development came out of the notion that programming in the large and programming in the small required different types of language.

The key goals of BPEL are:

1. Define business processes that interact with external entities through Web Service operations.
2. Define business processes using an XML based language
3. Define a set of Web service orchestration concepts that are meant to be used by both the external (abstract) and internal (executable) views of a business process
4. Provide data manipulation functions for the simple manipulation of data needed to define process data and control flow.
5. Provide both hierarchical and graph-like control regimes.
6. Support an identification mechanism for process instances.
7. Support the implicit creation and termination of process instances as the basic lifecycle mechanism.
8. Define a long-running transaction model that is based on proven techniques.
9. Use Web Services as the model for process decomposition and assembly.
10. Build on Web services standards (approved and proposed).

BPEL is still a work in progress with the WS-BPEL 2.0 Specification having recently entered public review (10 September 2006). BPEL and SOA are inextricably linked in that a Service Oriented Architecture (SOA) represents a collection of best practices principles and patterns related to service-aware, enterprise-level, distributed computing. The importance of the relationship between BPEL and SOA can be seen in the fact that SOA standardization efforts at OASIS focus on workflows, translation coordination, orchestration, collaboration, loose coupling, business process modelling, and other concepts that support agile computing.

3.3.7 REST (Representational state transfer)

Representational State Transfer (REST) is a software architectural style for distributed hypermedia systems like the World Wide Web. The term originated in a 2000 doctoral dissertation about the web written by Roy Fielding, one of the principal authors of the HTTP protocol specification, and has quickly passed into widespread use in the networking community. REST strictly refers to a collection of architectural principles (described below). It provides a set of architectural constraints that, when applied as a whole, emphasises scalability of component interactions, generality of interfaces, independent deployment of components, and intermediate components to reduce interaction latency, enforce security, and encapsulate legacy systems. The term is also often used in a looser sense to describe any simple interface that uses XML (or YAML, JSON, plain text) over HTTP without an additional messaging layer such as SOAP. These two meanings can conflict as well as overlap. It is possible to design any large software system in accordance with Fielding's REST architectural style without using the HTTP protocol and without interacting with the World Wide Web. It is also possible to design simple XML+HTTP interfaces that do not conform to REST principles, and instead follow a RPC model. The two different uses of the term REST cause some confusion in technical discussions.

The universal syntax supports the use of hypermedia to navigate from a simple starting point such as a home page. Representations in a hypermedia REST system are often HTML or XML documents that contain both information and links to other resources; as a result, it is often possible to navigate from one REST resource to many others, simply by following links, without requiring the use of registries or other additional infrastructure.

REST advocates claim that REST:

- Provides improved response times and server loading characteristics due to support for caching
• Improves server scalability by reducing the need to maintain communication state. This means that different servers can handle initial and subsequent requests
• Requires less client-side software to be written than other approaches, because a single browser can access any application and any resource
• Depends on less vendor software than mechanisms that layer additional messaging frameworks on top of HTTP
• Provides equivalent functionality when compared to alternative approaches to communication
• Provides better long-term compatibility and evolvability characteristics than RPC. This is due to:
  • The capability of document types such as HTML to evolve without breaking backwards- or forwards- compatibility, and
  • The ability of resources to add support for new content types as they are defined without dropping or reducing support for older content types.

REST detractors note the lack of tool support and the scarcity of truly RESTful applications deployed on the web of today. Some claim that REST is applicable to GET, but unproven for other state transfer operations such as PUT. POST is often considered the only necessary client-to-server state transfer operation, and is treated as a mechanism to tunnel arbitrary method invocations across HTTP.

Turning back to the layered architecture of Web Services, the following subchapter discusses the responsibility of each layer.

### 3.4 The Web Services Architecture

To perform the three operations of publish, find and bind in an interoperable manner, there be a Web Services stack that embraces standards at each level. Figure 9 shows a conceptual Web Services stack [7]. The upper layers build upon the capabilities provided by the lower layers. The vertical towers represent requirements that must be addressed at every level of the stack. The text on the left represents standard technologies that apply at that layer of the stack.

![Layered architecture of Web Services](image)

**Figure 9 Layered architecture of Web Services**

The explanation of the responsibilities of each layer refers to the most important technologies used in Web Services, briefly introduced in Ch. 3.3.
3.5  Layer responsibilities in the Web Services architecture

3.5.1  Network Layer

The foundation of the Web Services stack is the network. Web Services must be network accessible to be invoked by a service requestor. Web Services that are publicly available on the Internet use commonly deployed network protocols. Because of its ubiquity, HTTP is the de-facto standard network protocol for Internet-available Web Services. Other Internet protocols can be supported, including SMTP and FTP. Intranet domains can use reliable messaging and call infrastructures like MQSeries, CORBA, and so on.

3.5.2  XML based messaging layer

The next layer, XML-based messaging, represents the use of XML as the basis for the messaging protocol. SOAP is the chosen XML messaging protocol for many reasons:

• It is the standardized enveloping mechanism for communicating document-centric messages and remote procedure calls using XML.

• It is simple; it is basically an HTTP POST with an XML envelope as payload.

• It is preferred over simple HTTP POST of XML because it defines a standard mechanism to incorporate orthogonal extensions to the message using SOAP headers and a standard encoding of operation or function.

• SOAP messages support the publish, find and bind operations in the Web Services architecture.

3.5.3  Service description layer

The service description layer is actually a stack of description documents. First, WSDL is the defacto standard for XML-based service description. This is the minimum standard service description necessary to support interoperable Web Services. WSDL defines the interface and mechanics of service interaction. Additional description is necessary to specify the business context, qualities of service and service-to-service relationships. The WSDL document can be complemented by other service description documents to describe these higher level aspects of the Web service. For example, business context is described using UDDI data structures in addition to the WSDL document. Service composition and flow are described in a Web Services Flow Language (WSFL) or Business Process Execution Language document.

3.5.4  Service publication layer

Any action that makes a WSDL document available to a service requestor, at any stage of the service requestor’s lifecycle, qualifies as service publication. The simplest, most static example at this layer is the service provider sending a WSDL document directly to a service requestor. This is called direct publication. E-mail is one vehicle for direct publication. Direct publication is useful for statically bound applications. Alternatively, the service provider can publish the WSDL document describing the service to a host local WSDL registry, private UDDI registry or the UDDI operator node.

3.5.5  Service Discovery Layer

Because a Web service cannot be discovered if it has not been published, service discovery depends upon service publication. The variety of discovery mechanisms at this layer parallels the set of publication mechanisms. Any mechanism that allows the service requestor to gain access to the service description and make it available to the application at runtime qualifies as service discovery. The simplest, most static example of discovery is static discovery wherein the service requestor retrieves a WSDL document from a local file. This is usually the WSDL document obtained through a direct publish or the results of a previous find operation. Alternatively, the
service can be discovered at design time or runtime using a local WSDL registry, a private UDDI registry or the UDDI operator node.

### 3.5.6 Service Flow layer

Because a Web service’s implementation is a software module, it is natural to produce Web Services by composing Web Services. A composition of Web Services could play one of several roles. Intra-enterprise Web Services might collaborate to present a single Web service interface to the public, or the Web Services from different enterprises might collaborate to perform machine-to-machine, business-to-business transactions. Alternatively, a workflow manager might call each Web service as it participates in a business process. The topmost layer, service flow, describes how service-to-service communications, collaborations, and flows are performed. WSFL or BPEL is commonly used to describe these interactions.

### 3.5.7 Vertical Function Sets

For a Web Services application to meet the stringent demands of today’s e-businesses, enterprise-class infrastructure must be supplied, including security, management and quality of service. These vertical towers must be addressed at each layer of the stack. The solutions at each layer can be independent of each other. More of these vertical towers will emerge as the Web Services paradigm is adopted and evolved [8].

Key initiatives in this area include:

1. **Web Services Transactions**: These specifications define mechanisms for transactional interoperability between Web services domains and provide a means to compose transactional qualities of service into Web services applications.

   The Web Services Transactions specifications describe an extensible coordination framework (WS-Coordination) and specific coordination types for:
   - Short duration, ACID transactions (WS-AtomicTransaction)
   - Longer running business transactions (WS-BusinessActivity)

2. **Web Services Reliability (WS-Reliability)** is a SOAP-based protocol for exchanging SOAP messages with guaranteed delivery, no duplicates, and guaranteed message ordering. WS-Reliability is defined as SOAP header extensions, and is independent of the underlying protocol. The purpose of WS-Reliability is to address reliable messaging requirements, which become critical, for example, when using Web Services in B2B applications. SOAP over HTTP is not sufficient when an application-level messaging protocol must also address reliability and security.

3. **The Web Services Security specification (WS-Security)** provides a set of mechanisms to help developers of Web Services secure SOAP message exchanges. WS-Security describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. These mechanisms can be used to accommodate a wide variety of security models and encryption technologies.
4 Web Services and Enterprises

4.1 Current State of the Art

Gartner [13] contend from their research that the pilot stage is over and the simple use of Web services in the enterprise is now a reality. But the simple use of standards isn't enough and should not be considered enough. Technology should never be used for the sake of the technology alone, nor should anyone assume that use of technology alone will fully realise the value of Web services.

Enterprises that use and derive value from basic Web services must explore how they can exploit the technology and new development paradigm in major projects and for greater impact. Users must protect themselves from the unique risks to which Web services expose them, and ensure that they gather the benefits that Web services provide. Gartner previously identified four stages through which enterprise IT departments, or groups within IT departments, pass on their way to effectively and thoroughly exploiting Web services:

- Education — Where preliminary training occurs
- Experimentation — Where the IT organisation pursues initial Web services development
- Execution — Where the IT organisation rationalises diverse Web services projects with other IT priorities and establishes control over Web services development and integration methodology
- Embrace — Where the IT department incorporates Web services into its development, management, maintenance and IT integration processes, enabling the exploitation of Web services' strengths and the mitigation of their weaknesses

Gartner believe that all but the most cautious IT departments should enter the third stage of this continuum, the execution stage, no later than the end of 2005. It is essential to move through the execution stage quickly because the stage represents a stalling of business benefits while IT practices catch up to Web services, which are easy to build, but not always easy to control.

The promise of Web services is still tied closely to the widely used specifications and genuine standards that make Web services effective. Selecting which of these standards to incorporate into an architectural strategy, and how firmly to require them (or shun them), is a critical decision for any IT department seeking to exploit Web services and their more-comprehensive kin, the service-oriented architecture (SOA).

Although developers continue to wrestle with the issues of when and how Simple Object Access Protocol (SOAP) and Web Services Description Language (WSDL) are most effective, and how they fit in with other elements of networked computing, the basic standards are clearly pervasive. Gartner believe that both must be considered fundamental to any Web services project. However, selecting from the larger family of advanced standards, informally known as "WS-splat" or "WS-star" (includes WS-Security and WS-Addressing), is not simple.

Choosing products or vendors is also difficult. Although the emerging Web services framework is not a product, it benefits from the use of various products in different aspects of Web services deployment, from conception to development, repository storage and deployment, as well as management and maintenance.

Addressing aspects of the framework that demand operational discipline is equally important. Failure is an all-too-real potential outcome of any transition to a more-contemporary IT strategy, and moving to an SOA catalysed by Web services is no exception. IT operations managers must consider strategies to address Web services, in part to ensure that Web services do not dwell outside the ordinary course of enterprise. Such strategies can make the difference in the effective deployment of Web services; without them, the enterprise risks impact on systems and service management. As always, the challenge is to adequately prepare.
4.1.1 Technical challenges remain

This section briefly describes two key challenges remaining.

4.1.1.1 Data Integration challenges

Qualitative surveys suggest that the number one enterprise IT concern today is data integration within the enterprise virtual organisation. (In this context, the term virtual organisation means a composition of federated business units, each contained within its own administrative domain). Designers and developers are faced with the challenge of deriving shared semantic meaning between heterogeneous data sets. Translation and conversion systems must be designed, assumptions around data often have to be made. Risk is further increased as knowledge on the source data models may be incomplete or unclear. Semantic Web is one initiative aimed at helping to resolve this issue, however, it is early days yet.

4.1.1.2 Component challenges

The emergent ‘Web 2.0’ model of web development can provide a much richer and more seamless user experience than the traditional full-page-refresh, but it poses some difficulties as well. At its fundamentals, the technology set employed entails using the browser's client-side scripting capabilities in conjunction with its Document Object Model to achieve a method of content delivery that was not entirely envisioned by the browser's designers. Issues arise with the need for javascript enablement and back-button / bookmark behaviours, however, these can be addressed or mitigated. However, they do point to the need for a revision to current browser design.

To summarise, the combination of data modelling technologies stemming from the Semantic Web domain and the maturation of loosely coupled, service-oriented, platform agnostic communications protocols is finally providing the infrastructure needed to integrate data across within and organisational boundaries notwithstanding the fact that significant challenges remain to be addressed.

4.2 Where is the Industry leading us?

Web Services can provide the technology underpinnings for distributed applications. When fully embraced, Web services and SOA change the design centre of solutions from application functionality to the business process that is, providing a more business-centric view of how things will be done to the benefit of the enterprise as a whole. In this way, Web services and SOA will enable tighter alignment between business and IT. When considering web services in the enterprise it is also useful to view them in the context of how they are likely to be used to drive innovation. This section provides a short overview on three major innovations, directly related to Web services and SOA that are likely to shape how enterprises are likely to exploit these technologies within and beyond the enterprise boundary.

- Web 2.0
- Software as a Service
- Event Driven Architecture

Each of the above innovations probably deserves to be studied in much more detail in order to better understand impacts and benefits etc. However, it is worth noting that their likelihood of success has been greatly enhanced by the cross-industry development of interoperability technologies such as Web Services and SOA.

Any one of these initiatives may have significant positive impact on the adoption of SOA and Web services, however, when you combine all three together one can only conclude that the interoperability landscape has changed forever. This technological shift looks likely to deliver a veritable tidal wave of change over the coming years.
4.2.1 Web 2.0

Web 2.0 is described as a new approach to doing business on the Internet and is a concept in continuous evolution. Only time will tell whether the companies applying this approach are truly profitable or "merely" trendy and useful. However, there are some interesting and potentially beneficial characteristics worth reviewing particularly in the context of the enterprise.

Like many important concepts, Web 2.0 doesn't have a hard boundary, but rather, a gravitational core. You can visualise Web 2.0 as a set of principles and practices that tie together a veritable solar system of sites that demonstrate some or all of those principles, at a varying distance from that core.

Web 2.0 offers a number of important capabilities that may be beneficial to the enterprise, however, at time of writing they should be treated with caution. Firstly, Web 2.0 techniques are seemingly unproven no matter how great the potential benefits. Furthermore there are the unknowns about unleashing democratising forces or even determining the outcome within the enterprise. And finally the level of effort, and actual cost of using these tools is generally unknown. Unlike the consumer web of today, enterprises still need a higher level of control than the current consumer Web 2.0 environment is inclined to provide.

The first premise of Web 2.0 is leveraging the power of the user. For example, fluid user tagging of content would be used instead of a centralised taxonomy. Web 2.0 entrepreneurs often consider the Long Tail, which is basically an observation that the vast majority of the attention market is based on niche content.

Web 2.0 is radically decentralised, as in the case of Bittorrent, a collaborative downloading co-operative that consumes a serious portion of all Internet traffic.

Blogs are considered web 2.0 instead of centralised "personal home pages", blogs let people easily post as much or as little as they want as rarely or as frequently as they want. Feed aggregators ensure that people only need to visit a single site to see all the feeds they subscribe to. Comments are enabled everywhere, allowing people to participate rather than passively consume content.

These may be viewed as using the web as a collaboration and information sharing platform. Using web services to create ‘mashups’ with a transparent caching and content delivery network that eases bandwidth congestion. Similar to Web 1.0 but with additional Web2.0 design patterns with client-side technologies, such as, AJAX providing a richer user experience.

All of the above can be translated into a small set of design patterns and a discrete technology set that essentially differentiates Web 1.0 applications from Web 2.0.

4.2.1.1 Web 2.0 Technologies

The core collection of technologies and web development techniques was christened AJAX, in a seminal essay by Jesse James Garrett. Ajax incorporates:

- standards-based presentation using XHTML or HTML and CSS
- dynamic display and interaction using the Document Object Model
- data interchange and manipulation using XML and XSLT
- asynchronous data retrieval using XMLHttpRequest (or Iframe)
- and ECMAScript (typically Javascript) binding everything together

The intent is to make web pages feel more responsive by exchanging small amounts of data with the server behind the scenes, so that the entire web page does not have to be reloaded each time the user makes a change. This is meant to increase the web page's interactivity, speed, and usability.

4.2.1.2 Applied to the Enterprise: some examples

A recent Gartner Group assessment states that Web 2.0 offers many opportunities for growth, but few enterprises will immediately adopt all aspects necessary for significant business impact. Web
2.0 applications are increasingly finding a strategic role in the enterprise. Recently Gartner advised financial services firms to use Web 2.0 applications such as wikis, podcasts and blogs in order to improve cross-enterprise collaboration and deliver personalised information to clients. The Gartner report noted that engaging with customers through call centres and via Internet campaigns is no longer sufficient and recommends that banks adopt Web 2.0 applications in order to provide personalised services to customers. A separate report by Genesys has said that call centres today are focusing on Generation Y's demands by leveraging online instant messaging, performing courtesy calls, and routing specific callers to specialist representatives.

4.2.2 Software as a Service (SaaS)

SaaS is a model of software delivery where the software company provides maintenance, daily technical operation, and support for the software provided to their client. One key characteristic is application delivery that typically is closer to a one-to-many model (single instance, multi-tenant architecture) than to a one-to-one model, including architecture, pricing, partnering, and management characteristics.

There are two types of SaaS providers. The first has often been referred to as an Application Service Provider (ASP) where a customer purchases and brings to a hosting company a copy of software, or the hosting many companies who access the software across the web.

The second type of SaaS provider offers what is often called software on-demand, where a company offers to customer's software specifically built for one-to-many hosting. One of the defining characteristics of internet era software is that it is delivered as a service, not as a product. Therefore, operations must become a core competency. Google's or Yahoo!'s expertise in product development must be matched by an expertise in daily operations. It's no accident that Google's system administration, networking, and load balancing techniques are perhaps even more closely guarded secrets than their search algorithms. Google's success at automating these processes is a key part of their cost advantage over competitors.

While SaaS isn't a new idea, the economic climate and rapid advancements in application development tools have combined to make today's SaaS providers more successful than their predecessors. Many companies now consider various IT functions and business applications commodities and not core competencies. This has made SaaS, essentially an outsourced application management business, more attractive today than ASPs and hosting services of the past.

SaaS alleviates the costs of traditional perpetual licensing fees and also eliminates the need for additional IT infrastructure investments to support new applications.

A variety of enabling technologies, such as service-oriented architecture (SOA) and Web services, permit SaaS to be more easily provisioned and metered based on actual usage levels. This means companies no longer have to pay for excess capacity. The bottom line? Lower total cost of ownership and quicker time-to-value.

As SaaS gains mainstream acceptance, it is becoming an important disruptive force in the software industry. And as long as the quality and reliability of SaaS solutions continues to improve, the appeal of SaaS will most likely grow.

4.2.3 Event Driven Architectures (EDA)

Event Driven Architecture (EDA) is also referred to as Real Time Enterprise (RTE) and Complex Event Processing (CEP). A modern real time enterprise is underpinned by an architecture that supports complex event processing delivered through content based routing of messages.

4.2.3.1 Overview of event-driven architecture

An event-driven architecture (EDA) defines a methodology for designing and implementing applications and systems in which events transmit between loosely coupled software components and services. An event-driven system is typically comprised of event consumers and event producers. Event consumers subscribe to an intermediary event manager, and event producers publish to this manager. When the event manager receives an event from a producer, the manager
forwards the event to the consumer. If the consumer is unavailable, the manager can store the event and try to forward it later. This method of event transmission is referred to in message-based systems as **store and forward**.

EDA is all about message routing. There are basically two methods of defining message routing: it can be predetermined – flow based – such as you might see in a traditional, client server type application. If it is not predetermined then it must be evaluated at runtime based on the content of the message or events that occur during the system operation, you might see this, for example, in an Email System. The later is required in EDA. It is of course quite possible (even perhaps quite normal) to see systems that are hybrids of these two. So flows are predetermined because they can be, where as others are not.

Internet transactions, business-to-business systems, peer-to-peer processes, and real-time workflows are too dynamic and too complex to be modelled by traditional sequential-processing methods. Therefore, the need for more sophisticated asynchronous processing techniques is quickly becoming apparent. To address these unpredictable environments, the current trend in systems architecture is service-oriented design and event-driven programming.

Responding to real-time changes and events in a timely manner is becoming one of the most important requirements for an enterprise framework. A service-oriented architecture (SOA) presents a dynamic runtime environment, where loose couplings between service providers and/or service consumers enable powerful and flexible component interactions. Service-oriented and event-driven architectures are natural fits for distributed systems since they share many of the same characteristics, such as modularity, loose-couplings, and adaptability.

**4.2.3.2 SOA and EDA together**

SOA describes modular and distributed software that offers well-described interfaces. Components can access each other independently by request/reply functions. SOA differs from the more-general client/server model in its definitive emphasis on loose coupling between software components and in its use of separately standing interfaces. Integration of components at runtime can be done by orchestration mechanisms ensuring the right processing of components to perform a task.

SOA differs from an event-driven architecture (EDA) and also from a monolithic architecture. In EDA, all participating software components are decoupled from each other and not linked by necessary reply functions. In monolithic applications all software components are designed to operate only in the initially intended context (that is, logically tightly coupled).

It is generally considered that Web services (WS are a standardised subset of an SOA). SOA 2.0 is a new concept that covers the integration of SOA and EDA. Within the SOA, services are software functions that are executed in reaction to a computing request.

Unlike a request/reply system, where callers must explicitly request information, an event-driven architecture (EDA) provides a mechanism for systems to respond dynamically as events occur. In an EDA, events are published by event producers, and event consumers receive events as they happen.

Business systems benefit from the features of both an SOA and an EDA, since an EDA can trigger event consumers as events happen and loosely coupled services can be quickly accessed and queried from those same consumers. For systems to be most responsive, they must be able to quickly determine the necessary actions when events are triggered.

**4.3 Emerging business requirements**

Business context: It is generally accepted that subscriber growth is stabilising, revenues are continuously decreasing with voice rapidly becoming a commodity. Operators must create new revenue streams, and extend/ differentiate their service offerings. The closed nature of service provider’s networks has necessitated the development of applications that rely on proprietary Application Programming Interfaces (APIs) – which make the porting of these applications to different platforms difficult at best and lowers the service provider’s return on investment.
Operators around the world are adapting to a profound technology and service evolution, based on Internet Protocol (IP), broadband, mobility and all sorts of net-centric applications for consumers and businesses. Competition is increasing and prices and margins are coming down in many cases. New opportunities and challenges are appearing, such as IPTV, hosted voice, location-based services, mobile Internet and IT services. Changes in products and automation are transforming carriers' operations, processes and skills requirements.

With the convergence of telecoms, media and IT, one can see the arrival of Digital Media. DSL and cable based-broadband networks are rapidly moving into triple and more recently quad play models, delivering voice, data and video services.

To respond to these new challenges and opportunities, operators must carry out a cost-effective transformation. Competitive pressures will dictate the time frame for this transformation and Operators will need to define what core business capabilities they should keep in-house and those they should out-source.

Furthermore, the race is on to own certain classes of core data: location, identity, calendaring of public events, product identifiers and namespaces. Telecoms Operators hold strong positions currently on location and identity.

### 4.3.1 Drivers

The consumer electronics market is pushing for changes driven by LCD and plasma screens, DVRs and home entertainment systems; thus increasing market demand for more and better network services and capacity.

New digital-based broadcasters are rapidly moving into the areas of broadband and telecommunications. It is claimed that Broadband TV (IPTV) could double the amount of revenue from the top 25% users of high speed broadband. By 2008, we expect video streaming technology to be mainstream and fully integrated into any broadband service.

Changes in the telecommunication and media markets are being forced upon the industry by the new emerging Internet companies such as Google, eBay/Skype, Yahoo!, Vonage, AOL, MSN, and Amazon. They are breaking down the old business models in the industry that are mainly built around monopolistic market structures.

Video-based services on broadband and interactive digital TV networks are becoming whole new areas for advertising opportunities. Personalised media and one-to-one communication will be the predominant advertising mode using Digital Media, generating billions of euros in advertising and commerce worldwide.

At the heart of a digital home is the technical concept known as the Media Centre. It is predicted that by 2015 90% of all households in the developed market will have a home media centre. One industry analyst firm claims the global market for VoD will grow to over 350 million households by 2010.

Cable TV operators, telecoms operators, consumer electronics and IT companies are all vying for the Media Centre business. Progress in this market will continue to evolve with more mass market developments expected from 2008 onwards. Mobile TV is becoming available in selected markets during 2006 and may reach mass market by 2008.

### 4.3.2 Responses and implications

In a converged telecommunication and media environment, the focus will move toward content and applications, Operators need to upgrade their operations support, customer care, billing, network operations, business intelligence and other related information systems to support new services and reduce costs. In recent years, increasingly integrated solutions are becoming available while speed to market, flexibility and integration of new advanced systems require open and evolving architectures.

Telecom Operators need to align their structure, staff, management models and processes with new services, changing revenue and agility requirements. This includes issues about centralisation,
standardisation, process management across organisational entities, the degree of sourcing and the role of partnerships. Operator transformation also offers opportunities for external IT service providers and business process outsourcing (BPO) companies.

Telecom web services will be the preferred way for service providers and network operators to deliver new services to enterprises, augmenting the inherent strengths of enterprise application servers based upon J2EE and .Net. Telecom web services will enable enterprises to access telecom network capabilities and value-added services and to create customised telecom services with the aid of the Operator. Enterprises will be the ultimate winners as they will be able to freely select service provider and services. They will also be able to build the telecom services they need instead of relying on limited offers from existing service providers.

One of the aims of SOA is to be able to support flexible and dynamically reconfigurable end-to-end business processes using interface based service descriptions. In this context services are autonomous platform-independent computational elements that can be described, published, discovered, orchestrated and programmed using standard protocols for the purpose of building networks of collaborating business applications distributed within and across organisational boundaries.

The move from today’s sizeable investments in ‘stand alone’ systems, such as, PBXs, ERP, and Work Force Automation, to cost efficient hosted or semi-hosted network based services, integrated using web services, will have a significant impact for enterprises going forward.

Service providers will also clearly benefit, as they will simple, standardised and network independent manner. Simply speaking, business partners can use the published services of an operator and be able to access network capabilities within their own business processes and operator’s, in turn, can use services provided by business partners within their own business processes.

4.3.3 Challenges and Critical Success Factors

There is a core set of critical success factors that must be satisfied to ensure the full business benefits are realised from services deployed within and beyond the organisation. They may be summarised as follows:

Business Environment factors:
1. Supply chain management including partner management
2. Revenue sharing
3. Customer Management
4. Service Management
5. Marketing to increasingly specialised segments

Network Environment and Terminal Devices factors:
1. Service assurance
2. Service delivery (see above)
3. Service enablers
4. User databases
5. Billing and Charging

The sum of the business and network elements that are needed to support the delivery of these new services is commonly referred to as the Service Layer. Most Telecoms Operators today are looking to the service layer as a key element in their overall strategy. SOA and Web Services are considered key enablers to interoperability within and beyond this layer.

The main goal of the Service Layer’s architecture is to provide support for the smooth deployment of an increasing number of applications and users, while giving the operator total control over operating costs.
At another level, there are a number of challenging issues that need to be addressed including:

1. service modelling and design methodologies
2. architectural approaches
3. service development
4. deployment and composition
5. programming and evolution of services and their supporting technologies
6. infrastructure

Last but not least the integration of existing systems is and will ever be an issue with the introduction of new technology.

4.3.4 Case of the legacy applications

Telecoms Operators, in general, own hundreds of applications structured into silos with little, or very tenuous, communication between them, implemented on different platforms, with different technologies and based on different programming skills. Today, this legacy must be transformed to provide more agility and convergence between the different products. In addition, the boundaries between OSS and BSS are blurring. The trend is toward using a real-time, component-based software architecture that follows SOA principles and EDA.

As the enterprises need flexibility in an ever changing world, new applications should be developed, existing applications modified and all applications need better integration. The existing applications cannot be simply dropped/ replaced for the sake of having better integrated, advanced applications. Furthermore the programming skill of the IT staff is constrained, and re-training of the staff is a high cost action. If the functionality within legacy applications can be presented as services, with well defined interfaces, but keeping inside the well tested old ones, this can lead to considerable cost savings.

4.3.5 Business objectives

Enterprises are looking to eliminate as much delay and business process friction as possible, and SOA is a path to do this. SOA enables more flexible and reusable services that may be reconfigured and augmented more swiftly than traditional construction of applications systems. Hence, SOA can accelerate time-to-business objective, resulting in better business agility.

Achieving this business agility is a key tenet of improved competitiveness. Enterprises are looking to SOA to maximize return by reducing complexity and cost of change. SOA can mitigate the risk of technological and business change since SOA platforms offer standards-based services that can be reused. SOA increases business agility and responsiveness by increasing reuse of components and services, reducing new code creation and associated cost. Finally, enterprises are looking to improve business performance including customer satisfaction and improved value chain execution.

SOA is an approach for building distributed systems that deliver application functionality as loosely-coupled services. SOA provides a standard way to represent and interact with application functionality by leveraging open standards. This is critical to improve interoperability and integration across an enterprise and value chain. Standards also reduce business process friction by enabling the reuse of services. Developers can create new applications from existing components more quickly than building functionality variations from scratch. SOA allows the developer to focus on application assembly which speeds time to implementation.

Successful service-oriented architectures, however, begins with process modelling and careful collaboration between IT architects and business stakeholders. Before enterprises deploy SOA on a large scale, getting these teams together in a disciplined planning exercise can unlock the business value. The true value of Service Oriented Architecture is achieved when Business Architecture is joined with IT Architecture.
4.3.6 Overall benefits of Service-Oriented Architecture

- Increased Revenue:
  - Creates new routes to market, new value from existing systems
- Increased Flexibility
  - Flexible business models enabled by increased granularity of IT processes, called ‘services’
  - Allows easier development and maintenance of large-scale, distributed applications and services involving unpredictable and/or asynchronous occurrences
- Increased speed and quality of development
  - Offers the potential to combine and reuse pre-built service components for rapid application development and deployment in response to market change
  - Promotes component and service reuse, therefore enabling a more agile and higher quality development environment
- Increased efficiency
  - Integrates historically separate systems, facilitates mergers and acquisitions of enterprises
  - Reduced cycle times and costs for external business partners by moving from manual to automated transactions
- Services
  - Offers the possibility of offering new services to customers without having to worry about the underlying IT infrastructure
- Decreased cost
  - Eliminates duplicate systems, build once and leverage
- Decreased risk
  - Improves visibility into business operations

4.4 Common Information Model in SOA

4.4.1 What is a single information model?

A single information model is a technology independent specification of the characteristics of a set of objects, and their relationships to other objects in a managed environment, with no reference to storage methods, access protocols, or specific type of repositories.

Such an information model defines for each data element a single way to represent it. It may be used to instantiate Data Models and it defines their structure.

4.4.2 Single information model and SOA?

In a SOA, everything is exposed as a service. The data manipulated by these services must be mastered by users. The only way to achieve this goal is to rely on a single information model.

Reduction to absurdity: in absence of a Single Information Model, in order to invoke a service, we are going to need to translate/modify/map the data to be compliant with the service interface. Thus, we need a mediator (who knows for each service and each data the mapping to be done) that will adapt the data to the service requirements. No more "find, bind, execute", but instead a "find, bind, use a mediator, execute".
4.4.3 Example

In case of

- a VoIP provisioning service, whose interface deals with 16 characters names
- a VoIP supervision service, whose interface deals with 8 characters names

Where must the translation rule be stored? How to be sure that the user monitored is the good one? The only solution is to use a mediator that will be in charge of the translation. So, the service will be used through a kind of proxy and not dynamically.

4.4.4 Comments

The EAI experience showed how difficult it was to define a canonical data-model in a specific context. This will be more complex in absence of a context (as the services will be used in different contexts: activation, after sales service...). However, a Single Information Model may be used to create the data models of new applications. These data models will extend the same information model and will be aligned together.

4.4.5 Conclusion

It's possible to create a SOA (compliant with the main principles: stateless services, loosely coupled services...) without a Single Information Model, but all the benefits of an SOA won't be reached. The paradigm "find, bind execute" is more relevant and easier to implement with a Single Information Model. However, this paradigm is still a concept far from being operational.

Yet, to rely on a single information model to instantiate the data models of new applications is a good way to ensure their alignments. It is an efficient way to implement a SOA.

4.5 ITU-T GTDD

Data definitions are needed to develop XML schemas for exchange of data about telecommunication networks and services between telecom operators. Global Telecommunications Data Dictionary (GTDD) is an initiative from ITU attempting to address these issues. The GTDD is a resource to be used to in developing standards for interoperable information interchange in telecommunications operations, administration, maintenance and provisioning (OAM&P) applications. Also, the GTDD is a resource for applications developers that may have a need to discover terminology used in information interchange.

The GTDD is closely related to the TM Forum Shared Information Model, which is a fundamental component for achieving interoperability in the NGOSS Technology Neutral Architecture.

The ideas of the GTDD have been shared between ATIS (Alliance for Telecommunications Industry Solutions), TMF (Telecommunication Management Forum) and ITU-T (The International Telecommunications Union – Telecommunications Sector). Study Group 4 (Telecommunication management) is in lead of this work.
5 Relevance and limitations to a Telco

5.1 SOA relevance to the business and operations of a Telco

The relevance of using SOA for OSS/BSS includes the answer to the following needs:

- To bring closer the business and IT viewpoints, bringing improvement at the conceptual level
- To reduce the gap between the business and the IT concepts, bringing improvement at the technical level
- To master the increasing complexity of the information systems
- To enhance the systems flexibility and agility

5.1.1 A new step in software engineering progress

SOA defines a way of analysing, designing and implementing the components of the information system. These components may be OSS or BSS ones, or components at the border of OSS/BSS. SOA applies to an intra enterprise context as to an inter enterprise one.

In that way, SOA is a new step in software engineering progress towards Separation of Concerns paradigm. It follows up the object and components approaches (Figure 10), bringing more loose-coupling, and recommending some implementation standard (rather than being a standard by itself), as web services.

![Figure 10 Positioning SOA services with regards to objects, components & process](image)

The following SOA properties are relevant for OSS/BSS as a whole:

- Service reusability: the services are designed to be reusable in several contexts
- Service contract: the services honour a common agreement on the services description exchanges (the contract may include Quality of Service)
• Service loose-coupling: the services are designed in such a way to minimize their mutual dependence
• Service abstraction: the services give no visibility to their environment about information not exposed by the contract
• Service composability: the services are designed so that they can participate as effective members of other service compositions/orchestration (called composite services), when required
• Service autonomy: the services can execute self-governance of all its processing
• Service statelessness: the services do not manage any request context (nevertheless, they can manage an internal state that is self specific)
• Service discoverability: the services provide a description that is clear enough to be found by a discovery mechanism

5.1.2 Some infrastructure products are available to implement an SOA

Some infrastructure products are available to implement an SOA: EAI enriched and standardised (WS-*) with a QoS component. Two categories may be distinguished:

• Complete solutions: IBM (WPS 6), BEA (WLI or Aqualogic), Oracle (Fusion), Microsoft (WCF), …
• Functionally light solutions: Sonic (Sonic ESB), Iona (Artix et Celtix), CapeClear (CapeClear ESB), …

There is no open source complete solution available.

5.1.3 Impacts on the enterprise

We identified the main impacts on the enterprise with its adoption of SOA as the following ones:

• At the business level, to stimulate
  • the enterprise agility
  • the closed loop enterprise (Figure 11)
  • the extended enterprise
• At the information system development level, to make the service development (versus application development) and usage emerge
• At the information system urbanism level, to define new urbanism rules: the difficulty is to identify the relevant services
• At the information system integration level, to standardize both intra and inter enterprise integration in the same way
• At the business process level, to promote some business process outsourcing
With these impacts, we also identified some new business perspectives:

- Service lodging for ASP services
- Service mediation
- Consulting

5.1.4 The relevance of SOA for OSS/BSS must be supported by adequate governance

Well-founded governance for making SOA relevant to OSS/BSS must meet the following principles:

- The consistencies of all the SOA initiatives must be guaranteed
- A SOA strategy must be defined and shared among all the involved actors. Every of these actors have contracts with their business partners whose roles\(^1\) are: access service provider (ASP), service providers, hosting, choreographer, and other operators. Moreover, the operator must generally manage several IS\(^2\) (Fixed one, Mobil one …) and one IS owns many services
- The introduction of SOA must be iterative and incremental
- Some (business, IT, technical) policies must lead the SOA deployment
- Some dedicated organisation entities must be responsible for the development, the deployment and the management of the services; there are four phases of a service life cycle: abstract\(^3\) service description, assignation of the operator role\(^1\), integration (choice

---

\(^1\) A role represents an atomic trade likely to be able to propose a viable autonomous economic model in a competing environment.

\(^2\) An Information System (IS) represents the whole of the services whose a role needs to ensure the economic equilibrium of its activity.

\(^3\) An abstract system is a system described in a behavioural language which can be simulated in a framework independent of all technology.
of a technological target, for example Websphere with EJB, etc, according to hosting technologies), and deployment

5.2 SOA experiment in Telco

5.2.1 Analysis of SOA applied to a Telco

1. We know that all are services (and processes) with SOA. Today operators may have several OSS/BSS separated in several business units and in several countries that cannot communicate well together. Suppose we broke these monolithic big applications which are OSS/BSS into a provisioning IS and a charging IS: in that case SOA will ensure interoperability between these IS services.

2. SOA advises that services are abstracted before deployment and implementation to a technology. This step would open in future with MDA, automatic code generation, offering choice in technology targets.

3. A contextual drawback: The SOA community and Telecoms Operators have not explored enough such an architecture for telecom and network services.

5.2.2 SOA to support the exchanges between information system and service platforms

The lifecycle of a commercial service provided by a Telco may be divided into several phases:

- The service building and deployment
- The service ordering and delivery (or provisioning)
- The service usage
- The service monitoring and trouble management
- The service billing
- The service retirement

During the service provisioning phase the commercial order created at the Customer Relationship Management (CRM) level is divided into services that are sent to the Service Configuration & Activation (SC/A) level. At this later level, the input services are divided into elementary or composite services, respectively provided by one or several service platforms.

To ensure better integration between information systems and service platforms, we propose to wrap the service platforms with a mediation layer providing adaptation mechanisms and functions.

This mediation layer is based on a SOA broker with web services (Figure 12) enriched with repositories and business specific functions.
5.2.3 VNO

Virtual Network Operator (VNO) is a service operator that does not have its own licensed spectrum (mobile only) and does not have the infrastructure to provide service to its customers (i.e., it does not own the network on which its voice and data traffic is carried). Instead, MVNOs lease wireless capacity from pre-existing mobile service providers and establish their own brand names different from the providers. MVNOs typically offer subscription-based voice and data service, and the customers are not doing business with underlying wireless provider but with the MVNO brand.

In the case of SOA the operation of a MVNO would be simplified, since the API towards the operator would be automatically discoverable. The typical operations of OSS and BSS could use SOA compliant interfaces, and thereby allow an operator to easily be a part of the virtual operators value chain.

5.2.4 IPTV

Recently the converged services that provide communication and broadcasting feature have been emerged. IPTV is the typical service of that. Actually, most voice and data service providers are deploying or planning to deploy IPTV services. Using IPTV, the service provider can provide various value-added services like caller ID (This service enables a user who is watching the TV receive a voice call identification on the TV screen and then provides the option for pausing, and recording the video stream for the duration of the call).

To implement such a service, the most important thing for the service provider is to be able to manage the multicast group and its membership, but no unified standard for this currently exists. Using SOA principles the network independence of the service is possible.

One example scenario is “Caller ID on IPTV”, and this description is taken from a Parlay-X contribution to the Prague Accelerator meeting in 2006 from ETRI in Korea.

Service scenario described in steps;

- A user is watching a TV program on his living room television when an incoming Phone Call arrives
• The user’s phone rings and since the television is “on”, the Caller-ID information appears on
  the TV Screen
• An options menu appears on the TV screen
• Using Cursor Controls on the Remote, Control the user highlights the desired action and
  presses OK on the remote control (Alternatively, the user could just answer the phone)
• When finished, the user continues watching the TV programme
• The Programme could automatically pause while the customer makes a decision and
  automatically un-pauses when OK is pressed

There are various requirements to implement this service that are, as follows:
• Perhaps most importantly, the third party must be aware that the users are watching the TV to
  generate a Caller ID on TV pop-up
• The called telephone number has a relationship with the IP address of the IPTV
• There is the proprietary protocol between Parlay X gateway and IPTV head-end
• The IPTV terminal (set-top box) has the function of processing SIP and IGMP and translating
  from the former into the later

Figure 13 shows an example using possible Parlay-X technology to implement the service in an
operator’s network.
6 Conclusions and Recommendations

SOA is a new and evolving technology, based upon tried and proven principles. It offers many benefits for most IT organisations and especially to telecoms operators. Despite the general disillusionment with respect to SOA in 2005, Gartner still expected the support for SOA to grow and to mature as a technology within ten years although many changes in user and vendor organisations and technologies are required before SOA reaches its full potential. In the longer term, Gartner believed that SOA has the potential to be transformational to a business. It is, however, an open question, whether this will become true, as in the Gartner’s Hype Cycle for Emerging Technologies in 2006 it has completely disappeared from the presented view of the Gartner group [14].

It must not be forgotten, however, that SOA is first of all not just a technology, but also a way of thinking and working, whose principles should have been applied since IT systems were first implemented and used. SOA should not be seen as a silver bullet, however, when properly implemented it can provide increased agility, flexibility, scalability, and a faster development cycle for business-level services. However, the full benefits of an SOA will only be realised when critical success factors, such as, common information model plus service modelling, design and deployment are also addressed.

An SOA is particularly well-suited to providing business-level software modularity and rapid reuse of software components. When combined together, an SOA with Web Services offers a cross-platform standards-based approach to component interoperability exploiting the loosely coupled model offered in the ‘find-bind & execute’ model. This provides a high level of flexibility and when coupled with business process engines, such as, BPEL, opens up the opportunity for composite applications composed of services combined in an ‘as needed’ model, either dynamically or statically.

Web-services technologies, which may make the service delivery platform for SOA, will still remain important architectural elements of future systems, be they called SOA or WEB 2.0, EDA or anything else. The capability of turning applications into “building blocks” that can be infinitely rearranged – and usually at great speed – gives a new way not only to reconfigure the services and the business, but to connect to suppliers, partners and customers.

It's strongly recommended that Telecoms Operators should not let these capabilities pass by without being adopted and used. The detailed recommendations can be found in a companion document: “Applying SOA to service delivery platforms”, Deliverable 2 of this project.
References


[3] Gartner's 2006 Emerging Technologies Hype Cycle,  
http://www.gartner.com/it/page.jsp?id=495475

[4] Service-oriented architecture, From Wikipedia, the free encyclopedia,  

[5] OASIS SOA Reference Model Technical Committee,  


