Tuomas Vanhanen

REQUIREMENTS AND A FRAMEWORK FOR BROKER BASED INTEGRATION IN SERVICE-ORIENTED ARCHITECTURE

Master's Thesis
4.12.2003

University of Jyväskylä
Department of Computer Science and Information Systems
Jyväskylä

ABSTRACT

Vanhanen, Tuomas Jooseppi

Requirements and a Framework for Broker Based Integration in Service-Oriented Architecture

Jyväskylä: University of Jyväskylä, 2003.

136 p.

Master's thesis

Service-Oriented Architecture (SOA) is a method for publishing services hosted by computer systems for the use of other computer systems. This method can be used to integrate applications and is therefore called Service-Oriented Integration (SOI). Integration brokers are a traditional method of integrating different kind of systems by sending messages from one system to another. This master's thesis gathers requirements for an integration broker in Service-Oriented Architecture and presents standards that can be used to build a SOI architecture using Web services. Web services are a method for creating a SOA with Internet technologies.

This research has conceptual, case-study and constructive elements. In the conceptual part of the research different integration architectures and methods for Service-Oriented Integration are presented. The case-study part gathers requirements for an integration broker from UPM-Kymmene Oyj. Finally in the constructive part a framework for an integration broker in SOA is presented. This framework is also the main result of this research.

KEYWORDS: Service-Oriented Architecture, SOA, Enterprise Application Integration, EAI, Web Services, Integration Broker, Framework

TABLE OF CONTENTS

1 INTRODUCTION

Organisations are becoming more dependent on their information systems at the same time as the systems are growing and getting more and more complex. There is also a growing need of integrating these systems, and this process is called ***Enterprise Application Integration*** (EAI). Linthicum (2000a, 3) describes EAI as the "unrestricted sharing of data and business process among any connected application and data sources in the enterprise".

The content of EAI has been transmuted from a simple integration of two applications into large-scale enterprise-wide integration. The first EAI solutions were based on simple messaging techniques, whereas now we have message brokers that contain various integration technologies; such as adapters, transformation possibilities, workflow support and technology for publishing and subscribing message topics. (Gawlick, 2001, 473)

The need for integration derives from poor architectural design and implementation of new systems without taking time to plan which platforms and applications should be used. Also mergers and acquisitions lead into situations, where different kinds of systems had to be integrated. For example these systems could be old legacy systems, ERP (Enterprise Resource Planning) systems, web applications and client/server applications. Different applications have overlapping data and business processes. Maintenance of the same business processes and the same data requires company resources. All integrated applications are affected when integration of different business processes and data takes place.

EAI brings several benefits, which include (Trumper, 2001, 48):

- Information systems are aligned with business vision, strategy and goals and enable,
- possibility to quickly and cost-effectively change systems and processes to support business need,
- ability to leverage investments in technology by integration of proprietary and packaged applications,
- increase of cost-effectiveness by reorganizing business processes and reducing system integration effort and
- identification of web services that an organisation can provide to its business partners and customers, during analysis of business processes.

During the last few years of discussion on EAI Web services have been identified often as a new way of integrating information systems.

> The term ***Web services*** refers to a group of closely related, emerging technologies that aim at turning the web into a collection of computational resources each with well-defined interface for its invocation. Web services are designed to be platform and language independent. (Elfatatry & Layzell 2003, 5)

Web services base on ***Service-Oriented Architecture*** (SOA), which is a loosely coupled architecture providing a model for dynamic integration. The philosophy in SOA is that systems provide services (e.g. currency exchange rates), which is remarkably similar to component model. Samtani and Sadhwani (2002a, 43) classify Web services as a technique of function and method integration, just like CORBA. Service-Oriented Architecture is not a new concept, and it is often defined with Web services, though one could argue that Service-Oriented Architecture is more of an extension of component model philosophy and of dynamic application integration (Tsalgatidou & Pilioura

2002, 136). Web services hold possibilities for similar re-usability and component design.

An idea similar to Service-Oriented Architecture is a view of selling software as a service, i.e. ***service-oriented model of software***, in which software functionality is delivered as a service. Every time functionality is needed, one identifies service elements, negotiates terms of use and uses the functionality. (Elfatary and Layzell 2003, 1 basing on Bennet, et al 2000 and Elfatary 2002).

When talking about integration with Service-Oriented Architecture, a term ***Service-Oriented Integration*** (SOI) has been used. In SOI components provide services to clients and this integration procedure is called Service-Oriented Integration (Brown et al 1992 according to Tombros et al. 1995, 3)

Service-Oriented Architecture is often described through its components. The following sections describe these components and their operations and define Web services more thoroughly.

1.1 Service-Oriented Architecture

Service Oriented Architecture supports a programming model, which makes it possible for components in a computer network to publish their services, and find and initialise other services hosted by other components. Typically these components interact in programming language in a non-dependent way. (Vivekanandan, et al, 2002, 1)

Service-Oriented Architecture consists of three different components (Figure 1): service provider, service registry or broker and service requestor. ***Service provider*** is a platform, which maintains the service and publishes (and deletes and updates) service description to the service registry. The ***service requestor*** is

the actual user of the service. It has a need that the service provider can fulfil. Service requestor finds a suitable service and its provider from the service broker. Then it initiates the interaction with the service provider by the service description information it got from the service broker. ***Service registry*** is a registry in which service providers publish their service descriptions. The service requestor uses the registry to find suitable service and the information needed to initialize it. (among others: Kreger 2001, 6-9 and Tsalgatidou & Pilioura 2002, 136-137, Samtani & Sadhwani 2002a, 44-45)

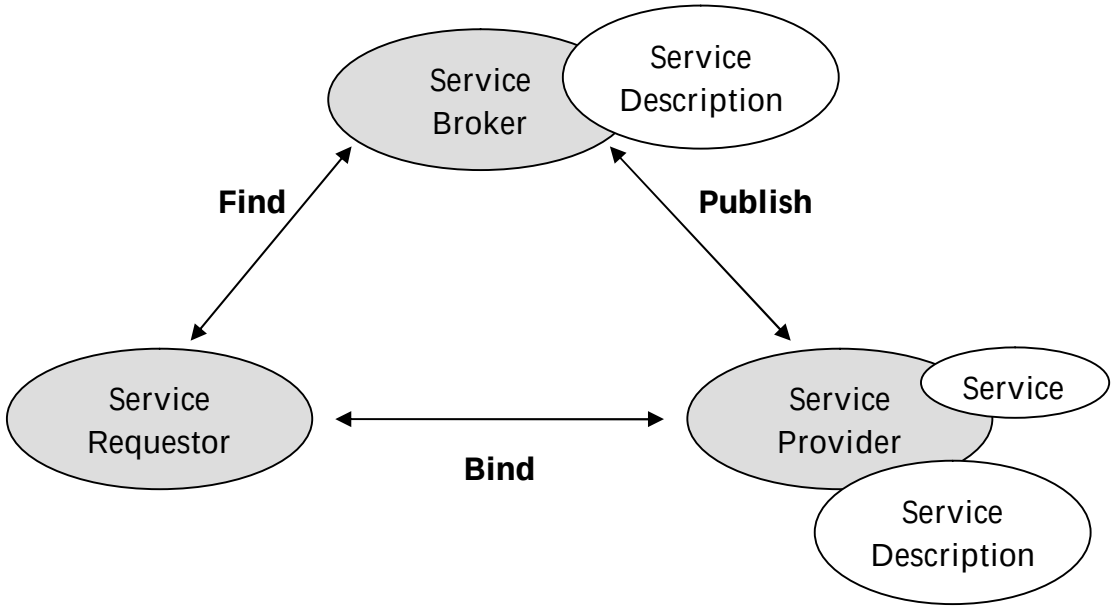


Figure 1. Service-Oriented Architecture (Simplified from Kreger 2001, 7)

Though Service-Oriented Architecture is often related with Web Services, it is a framework that does not require any specific technology. For example, it can use any network protocol from HTTP, SMTP, to FTP or CORBA RMI/IIOP and MQSeries messaging infrastructure. Also the document payload (e.g. EDI) can vary, though SOAP messages are the most common implementation. (Kreger, 2001, 10)

1.2 Integration Brokers

This master's thesis concerns the usage of integration broker technology with Service-Oriented Architecture. An integration broker can broker information (i.e. messages) between one or more target entities (e.g. networks, middleware and systems) regardless how this information is presented or accessed (Linthicum, 2000a, 291). Integration brokers are used to send messages in reliable manner from one application to another. Brokers also are able to convert the message format to the format that the recipient supports.

The use of an integration broker in SOI could bring several benefits: When a service provider changes the message format it uses (e.g. version update), a broker could transform incompatible older message versions to fit the new message format. Brokers also provide a reliable messaging infrastructure and are a logical place for holding service descriptions, since integration brokers already contain routing and transformation rules. Also brokers could be used to work as a gateway between different networks (see section 2.1.3). Integration brokers were also chosen because they probably are the most common EAI engine. In addition the case-study company currently uses this technology. Case-study and the case-study company are presented more thoroughly in sections 1.4 and 5.1.

Integration broker builds on top of traditional middleware such as Message Oriented Middleware; this is why integration brokers are also known as Message Brokers. Figure 2 describes how a message broker can integrate different types of systems and key components with a message broker. Message broker can provide services like message transformation, routing and a rules engine. Adapters and Application Interfaces (APIs) are used so that the broker

is able to communicate with different types of applications, middleware and data sources. (Linthicum, 2000a, 291-296 and Samtani & Sadhwani 2002a, 71)

The message transformation engine contains a dictionary of how each application understands the information and transforms the message suitable for the receiving application. The rules engine makes it possible to create rules for processing and distributing the messages. It recognises a message and makes it possible to transform it and distribute it to the right recipient(s) in a correct format. The intelligent routing service, also known as flow control and content base routing**,** identifies a message coming from a source application and routes it to the correct target application. The routing service builds on top of the rules engine and message transformation layer, it analysis the message and once it has recognized it, it can be transformed accordingly and routed to the right recipient. **(**Linthicum, 2000a, 297-303) Integration broker includes several other services, which are covered in section 4.1.

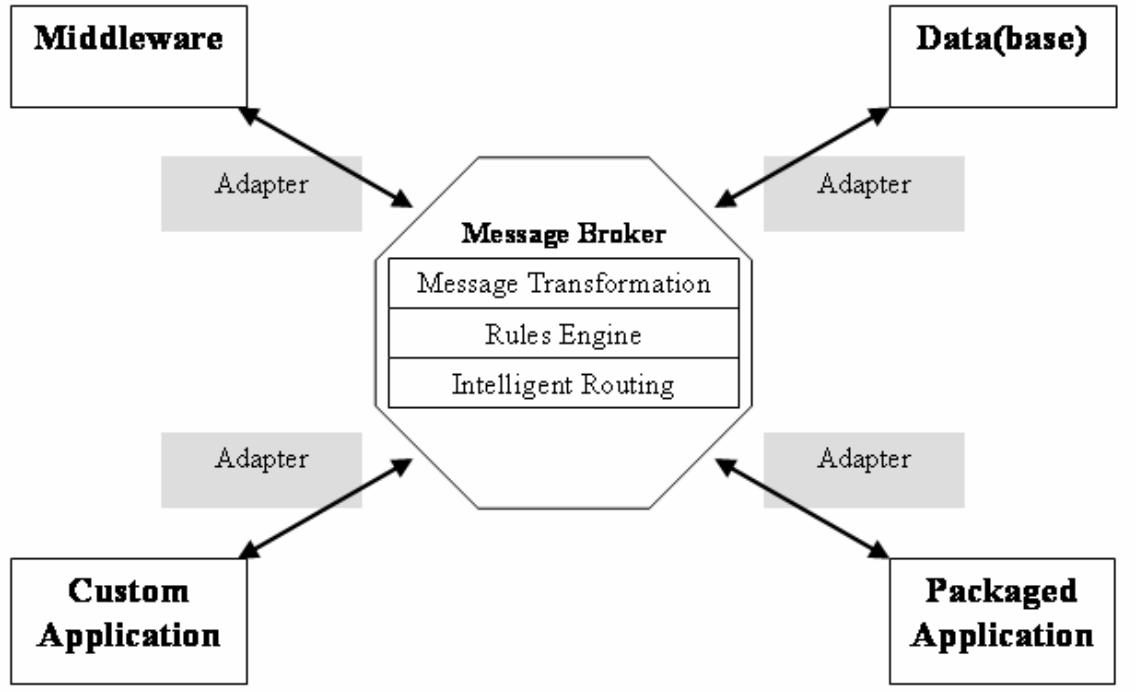


Figure 2. Message broker (adapted from Linthicum 2000a, 294 and Samtani & Sadhwani 2002a, 73)

1.3 Defining Research Issues

The use of Web services in Service-Oriented Architecture together with an integration broker is one of the areas which has lacked research. The purpose of this thesis is to research what kind of features an integration broker supporting Service-Oriented Architecture should have and to describe standards that help building a broker solution fulfilling these requirements. This research attempts to describe requirements for both intra-organisational and inter-organisational integration.

This master's thesis describes the nature Service-Oriented Architecture and Web services. It also presents different integration and Web services architectures. The role of the broker in SOA is illustrated by presenting requirements for an integration broker; these are gathered both from literature and by interviews in a case company UPM-Kymmene. Basing on the requirements and existing Web services architectures this master's thesis presents a general framework for an integration broker supporting Service-Oriented Architecture. The purpose is to create an entity, from standards related to Web services, which fulfils the set requirements for an integration architecture. One of the areas of consideration is which specifications will be strong in the future and become industry wide standards.

Research issues:

- What requirements can be found for an integration broker supporting Service-Oriented Architecture?
- What standards can be used when building an integration broker supporting Service-Oriented Architecture?

- What would a general integration broker architecture be like, when it supports Service-Oriented Architecture and which standards can be used to build this architecture?

The results of this thesis can be used when an organisation is planning to start using Service-Oriented Architecture in its integration infrastructure. This thesis will help to plan this infrastructure. The presented framework can be used when evaluating different integration brokers and integration architecture and the characteristics of the entities belonging to this infrastructure.

1.4 Research Methodology

This thesis is partly conceptual and partly a case-study. It is also constructive, since the purpose is to present a new framework of an integration broker. The results of the case-study are used to find requirements for an integration broker and to evaluate the presented framework. Still, most of the material gathered for this thesis will be from the literature.

The interviews are focused interviews (Hirsijärvi & Hurme 2001, 47-48). The interviewees were asked to prepare use cases of situations in which they would believe Service-Oriented Architecture would be useful. These use cases were studied together by the interviewee and interviewer to find out architectural requirements for the integration broker. Use cases were used for requirements gathering for example by W3C work group (2002b). The interviewees were selected by their knowledge of SOA and position in the case company UPM-Kymmene Oyj. Interviews took place in Helsinki, Kuusankoski and Lappeenranta during August 8-21 2003. UPM-Kymmene, with a turnover of €10.5, is the world's largest manufacturer of printing papers. The UPM-Kymmene Group has about 35,000 employees and has production in 17

countries. (UPM-Kymmene 2003, 7) The case-study is described in more detail in section 5.1.

The framework was build to fulfil the gathered requirements and by using the other Web services architectures as a basis. The ready framework is evaluated in the end of this thesis and it is discussed whether based on this case-study one can present a general architecture of an integration broker supporting SOA. The validation of the framework to other organisations is left to be a later research issue.

1.5 Defining the subject

The technologies covered in this thesis are covered in a general level, considering their potential and restrictions. When covering component technologies, like CORBA, EJB and DCOM, they are not described in very deep manner. They are mostly presented to provide a holistic view of possible payload methods in Service-Oriented Architecture.

The purpose of this thesis is not to present any code-examples, nor to present how different technologies are used in practice. This thesis concentrates to integration broker technologies, but also other architectural solutions are presented. This research does not try to prove whether to use or not an integration broker based integration architecture. This master's thesis presents a framework of an integration architecture, which includes the integration broker.

Chapter 2 in this thesis describes different integration architectures and methods that support Service-Oriented Architecture. It also describes some related technologies in an overall way. In Chapter 3 is presented several Web service architectures and standards. Chapter 4 contains requirements for an integration broker in Service-Oriented Architecture found in literature and in

the summary of these requirements the requirements are categorized and this categorisation is used in Chapter 5 to present the requirements, which were found during the interviews. Chapter 5 also summarizes all the requirements. Chapter 6 presents the framework for broker based integration in service-oriented architecture. Chapter 7 is the discussion and Chapter 8 is the conclusions.

Service-Oriented Integration can be achieved in many different ways. The following chapter describes different integration architectures and technologies for creating Service-Oriented Integration.

2 AN OVERVIEW OF ARCHITECTURES AND METHODS SUPPORTING SOA

The purpose of this chapter is to give a general overview of different integration architectures and technologies. First in section 6 are covered such architectures as point-to-point and many-to-many architecture. These are not covered in a very detailed manner, the main purpose is to present the main strengths and weaknesses of an integration architecture when using (or not using) a message broker. Section 6 also describes different components of an integration architecture, such as adapters and gateways, which are a part of the framework presented in chapter 6.

Services can be provided in many different technologies, with Web services, Remote Procedure Calls (RPC), etc. In section 2.2 it is presented different methods of providing a service. In section 2.3 these technologies are discussed and it is argued why this master's thesis uses Web services as the technology for Service-Oriented Integration.

Gold-Bergstein (1999) has defined six levels of application integration. Following list aims to present the topics covered in this chapter regarding Gold-Bergstein's division. This presented division is not used further in this research, it only outlines the aspects of application integration and that this section tries to cover all these levels:

1. Platform integration, which uses heterogeneous hardware and uses such technologies as Remote Procedure Calls (RPC) and Object Request Brokers (ORB). These are described in section 2.2.1.
2. Data integration with the use of database gateways or data warehousing technologies. The database gateways provide an access to heterogeneous

data sources with SQL (Structured Query Language). These do not directly provide services to SOA, they require some kind of a front-end, an adapter or a broker capable of performing SQL queries and therefore is not covered in this chapter.

3. Component integration, which means that an existing system is wrapped e.g. behind an application server. The application server can then use any suitable technology (e.g. RPC or Web service) of hosting the services, this layer is mainly discussed when describing adapters (section 2.1.4) and platforms for running Web services, in section 4.7.

4. Application integration with the use of adapters, message brokers and transformation rules. Adapters are discussed in sections 2.1.4 and 4.7.

5. Process integration, which provides a high level abstraction for integration purposes. Process integration tools are used to build, monitor and change business processes through a graphical user interface. This would be part of broker's functionality, and therefore is not covered in this chapter. Web service choreography methods are described in chapter 3.

6. Business-to-business integration through EDI and XML together with integration of supply chains and use of online trading brokers. Again B2B can be achieved with several technologies, of which EDI and XML are very popular. EDI's successor ebXML is described in section 2.2.4 and section 2.2.3 presents Web Services, which is an XML based method of making remote procedure calls.

2.1 Integration Architectures and Techniques

Johannesson et al. (2000) and Linthicum (2000a, 132-139) have described divisions of integration architectures. Both present a point-to-point architecture, which is described in section 2.1.1. Linthicum's many-to-many architecture is divided by Johannesson et al into broker architecture and process broker architecture. The latter one is a continuation from work flow management systems and contains sophisticated methods for process integration. Many-to-many architecture is described in section 2.1.2.

2.1.1 Point-to-Point Architecture

In ***point-to-point*** architecture every application is connected to each other directly (see Figure 3). This solution can be used with a small number of applications, because when adding another application into the architecture often requires integration with all the other applications. This increases dramatically the amount of connections needed. (Johannesson et al. 2000)

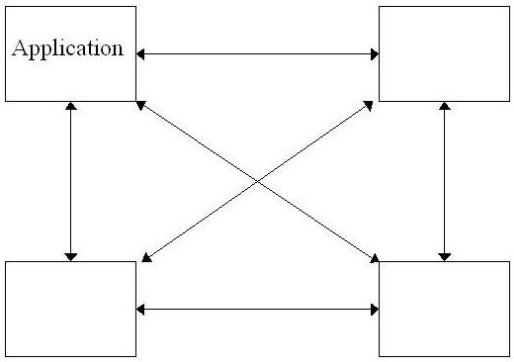


Figure 3. Point-to-Point Integration (Johannesson et al. 2000, 4)

Point-to-point architecture's strength is its simplicity. The EAI architect does not have to develop complex ways of integrating several different kinds of applications. Unfortunately point-to-point integration has several weak points, mainly the increasing complexity as the amount of integrations increase. It

cannot house application logic or change the messages which are transmitted, though it can use Message Oriented Middleware, other middleware or Remote Procedure Calls for communication. Therefore many-to-many architecture is considered as a best fit for EAI. (Linthicum 2000a, 132-134)

2.1.2 Many-to-many Architecture

The problems presented with point-to-point architecture can be mastered with a ***many-to-many architecture*** by using message broker technology. In many-to-many architecture a central message broker is used to integrate several applications at the same time (see Figure 4). In this way the amount of interfaces can be reduced and building and updating the integrations is easier. (Johannesson et al 2000. Johannesson & Perjons 2001)

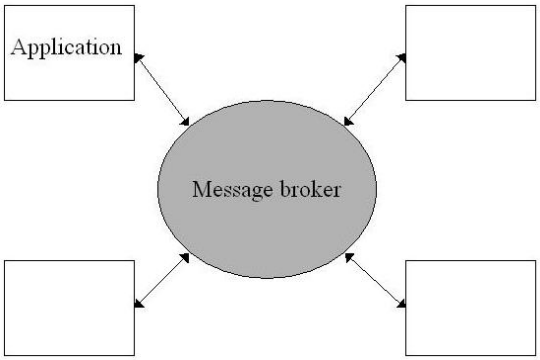


Figure 4. Integration with a message broker (Johannesson et al. 2000, 4)

The problem of message broker architecture is that the architecture can become quite complex when connecting several applications together. Though, current middleware products are getting better managing complex integrations. (Linthicum 2000a, 134-135) However, since the amount of interfaces for one application can be reduced, the management of interfaces is easier. If one application changes format, then the change has to be implemented only in the connection between the application and the broker. Also, the message broker

has tools for applying format conversions, so the change should be easier to manage. (Johannesson & Perjons 2001)

2.1.3 Gateway Access

Dictionary (Editors of the American Heritage 2000) describes ***gateway*** as "Software or hardware that enables communication between computer networks that use different communications protocols". Gateways can also be used for example to enable communication between different component models, e.g. DCOM and CORBA (Bechini et al 2002). This thesis presents gateway as a concentration point of inbound and outbound access from/to other networks. Such gateways can be, for example, between Wireless Application Protocol device and the Internet, see Figure 5 (Wap forum 2002, 8). The gateway transforms the protocol used in WAP environment (Bearer) to fit the Internet (IP). The gateway also supports encryption (e.g. SSL). Note that the newest WAP versions support HTTP and do not require transformation anymore (Wap forum 2002, 8).

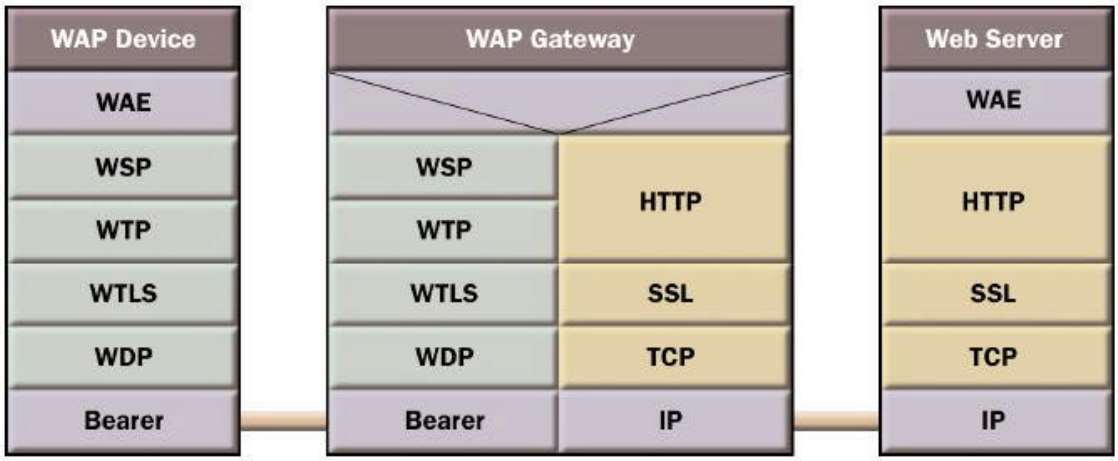


Figure 5. WAP Gateway (Wap forum 2002, 8)

In EAI gateways can be used to bridge communication between the broker and different networks like the Internet, mobile networks (WAP), partners using

leased lines, etc. Gateways can be used, for example, to transform Web service communication from HTTPR to Message Oriented Middleware. This situation could exist, when a partner is communicating over the Internet (HTTPR) and transmitting the message to the final recipient, which is in the company's Intranet (MOM).

2.1.4 Adapters

Adapters are layers between the source/target application's interface and the message broker interface. The adapter contains a set of libraries, which are used to map the differences between the two interfaces. Adapters ease the integration of the application to the message broker by hiding the complexity of the mapping from the developers. (Linthicum 2000a, 309)

Linthicum (2000a, 309) divides adapters to thin and thick adapters. Thin adapters simply map the target or source application's interface to the common interface which the broker supports. Thick adapters on the other hand have more functionality. Unlike in thin adapters, thick adapters contain management tools for changing the mappings without touching the source code. Linthicum (2000a, 309) also describes, that adapters can either be centralized or distributed. Centralized adapters run with the message broker. These are typically thin adapters that change the message broker's API to fit the source or target application. Distributed adapters are often thick adapters that run at the integrated application or at the broker. When the adapter is running at the application-end, it adds more functionality to the system, like capability to capture events.

Hildreth (2000) has a similar division as Linthicum. She divides adapters to static and intelligent adapters. Static adapters are traditional or standard adapters, which simply connect the application to another application or to the

integration broker, without any special features. Intelligent adapters on the other hand can perform dynamic tasks, of which many are often performed by an integration broker. Such tasks are data transformation, event routing, queuing. An intelligent adapter must provide an event trigger, do transformations and have some workflow capabilities.

The Intelligent adapters Hildreth presents take some of the responsibility away from the integration broker. This is not a good idea from the management point of view, since the mapping and transformation rules do not exist in only one point. The intelligent adapter Hildreth present might be useful in point-to-point connections, but it might increase the maintenance workload when working with integration brokers.

Kuebler and Eibach (2000) describe how the integration is performed when using WebSphere Application Server and Web service adapters (see Figure 6).

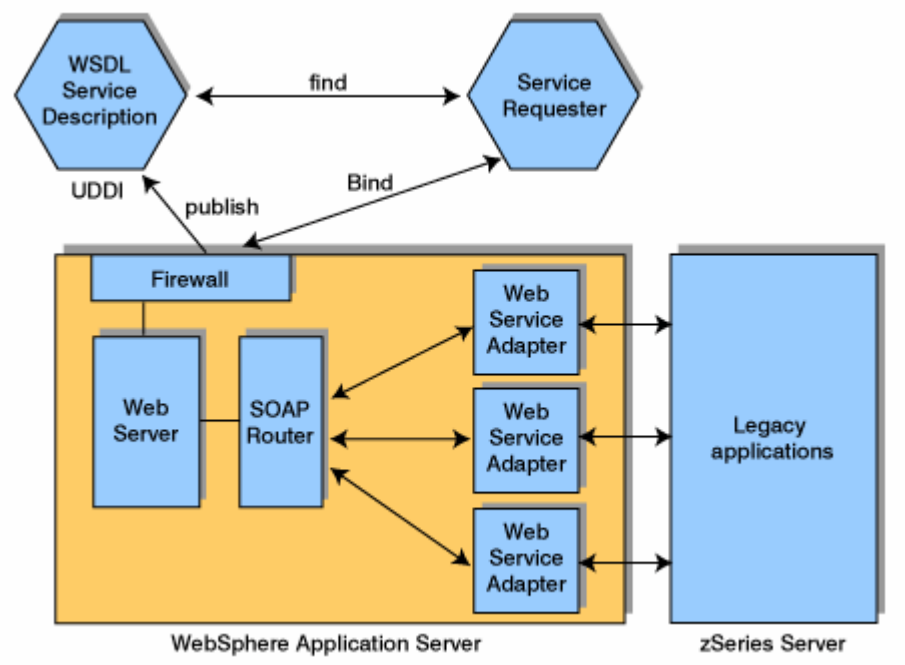


Figure 6. Connecting legacy systems with Web Service Adapters (Kuebler & Eibach 2000)

In this example, after the service has been found, the SOAP request is sent by the service requestor with HTTP POST through the firewall to the Web server. Web server analyses the HTTP header and finds as a part of it a Uniform Resource Name (URN) the name of the SOAP router component, to which it passes the message. The SOAP router then analyses the HTTP header and finds the location of the correct Web service adapter, to whom it then transmits the message.

2.2 Methods of Service-Oriented Integration

This section describes some methods for Service-Oriented Integration. Most of these are alternatives to Web services and are typical ways of integrating systems. Remote Procedure Calls are covered first, then Component models, Web services and ebXML.

2.2.1 Remote Procedure Calls

Remote Procedure Call is a method of invoking a procedure on the server computer. The client is suspended while it waits for the reply from the server; this diminishes the performance of that application. Also RPC requires quite a lot of network bandwidth. Distributed object technologies leverage RPC to provide object to object communication. Distributed objects are covered in section 2.2.2. (Birrell & Nelson 1984. Linthicum 2000a, 164-165)

The Remote Procedure Call environment consists of the client (user), client stub, RPC runtime (client), server, server stub and RPC runtime (server). Figure 7 illustrates these components and their relationships. When the client makes a remote call, it actually makes a normal local call invoking the corresponding procedure in the client stub. The client stub places the client's specifications and arguments into one or more packages and requests the RPC Runtime to

transmit these to the server machine. When the RPC Runtime at the server machine receives the RPC packages it passes them to the server stub, which unpacks the packages and calls locally the server. The reply is routed in a similar way to the waiting client. The RPC Runtime is responsible for retransmissions, acknowledgements, packet routing and encryption. (Birrell & Nelson 1984)

Open Software Foundation's Distributed Computing Environment (DCE) is a middleware solution for making diverse computers function as a single virtual system. The core of DCE is basic RPC mechanism. Developers can use DCE to tie different systems around the organisation together, though many organisations are transforming from DCE to Message-Oriented Middleware. (Linthicum 2000a, 165-166)

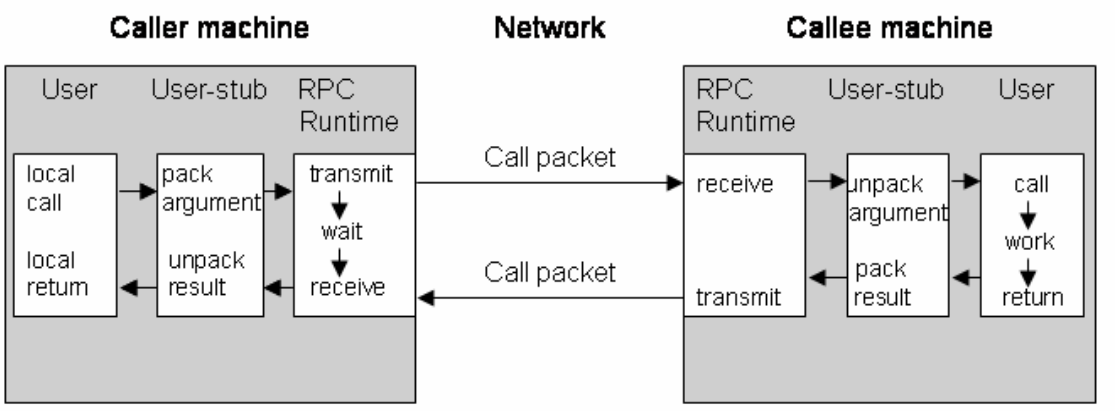


Figure 7. Components of the Remote Procedure Call system (Birrell & Nelson 1984, 44)

2.2.2 Component models

This section describes briefly component models which can be used to host and call services. Such technologies are CORBA, Microsoft DCOM and Java's EJBs.

CORBA

CORBA (Common Object Request Broker Architecture) is Object Management Group's (OMG) vendor-independent architecture and infrastructure for interoperable communication between different computer applications (Object Management Group 2002). CORBA applications consist of objects, which combine data and functionality and are individual units of running software. The capabilities and interface of the object are described with Interface Description Language (IDL). IDL interface definitions are independent of programming languages, but do map to the popular languages with OMG standard mappings. OMG supports for example C, C++, Java and COBOL. (Object Management Group 2002) IDL defines the input parameters, return values of the operations and any exceptions these operations may raise (Gokhale et al 2002).

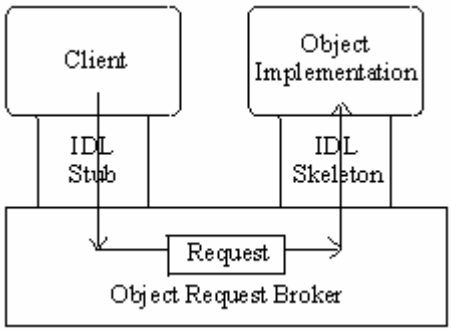


Figure 8. Simplified CORBA Architecture (Object Management Group 2002)

Figure 8 describes a simplified invoking process. The interaction between the client and object implementation is handled by the Object Request Broker (ORB) (Object Management Group 2001). Instead of calling directly the object implementation, client calls an IDL stub. It and IDL skeleton work as proxies for the client and the server and are generated by the ORB basing on the IDL. The ORB converts (maps) a request to fit the IDLs and this is the reason why clients and objects can communicate even if they are programmed with different

languages. When calling an object hosted by another Object Request Broker, the two ORBs communicate using Internet Inter-ORB Protocol (IIOP). (Object Management Group 2002) CORBA communication requires that the client knows exactly which object it is going to invoke and what kind of parameters it has to use.

DCOM

Originating from Microsoft's operating systems DCOM is nowadays managed by independent ActiveX Consortium. Though DCOM supports some Unix platforms, it is still mainly used in Windows environments. DCOM stands for Distributed Component Object Model and it is an extension to COM and COM+, which did not have the ability to communicate with COM-enabled ORBs. (Linthicum 2000a, 186-189. Microsoft 1996)

In DCOM architecture (see Figure 9) DCOM network protocol is used to communicate between the two ORBs, also known as automation servers. These automation servers were also known in COM-architecture. The client invokes the service of an automation client (in the image COM run-time) through a common COM-interface. (Linthicum 2000a, 186-187)

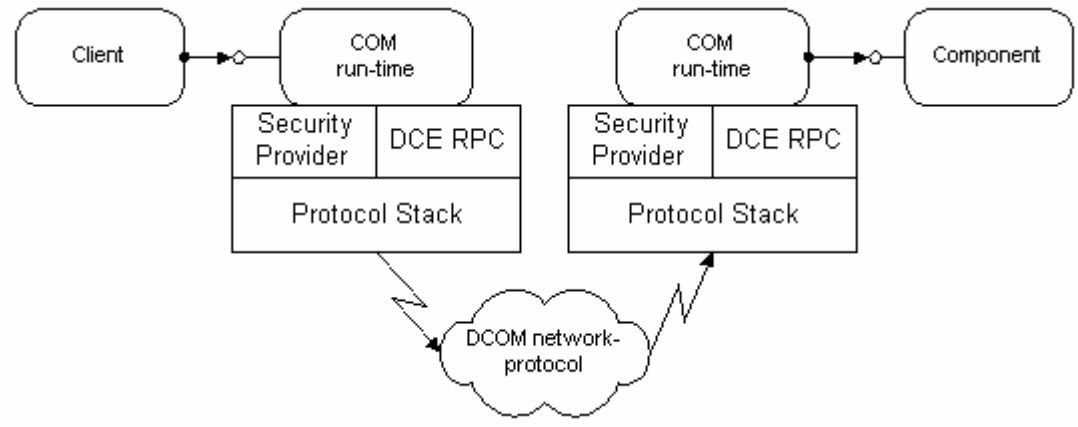


Figure 9. DCOM Architecture (Microsoft 1996)

The DCOM wire-protocol is based on COM's DCE RPC (Distributed Computing Environment). DCE RPC is a standard for converting in-memory data structures and parameters into network packets. DCOM's security providers are OS based security services which make the authentication possible. The security services are based on a central user directory, which contains such information as user names, passwords and public keys. (Microsoft 1996) On the wire level the DCOM protocol, known as Object RPC (ORPC) uses standard RPC packets with additional DCOM-specific information (Microsoft 1997).

EJB

The Enterprise JavaBeans (EJB) is an architecture for component based computing and for building and deploying distributed business applications (Sun Microsystems 2003, 27). The core of the EJB architecture is an EJB container. The EJB container hosts the components, also known as enterprise beans. Single container can host several enterprise beans and for each bean it also hosts a home interface, which enables the client to create, find and remove entity objects that belong to the entity bean. In addition, the home interface can provide business methods that are specific for that particular bean object. (Sun Microsystems 2003, 123-124) The EJB standard enables interoperability between containers produced by different vendors. EJBs use RMI (Remote Method Invocation) protocol for communication between clients and components, but they also support CORBA's IIOP, which makes them CORBA compliant. (Lang 2003, 96) The proposed EJB specification (version 2.1.) requires the EJB implementations to support Web service and its technologies WSDL, SOAP and HTTP. The EJB specification also contains among others transaction and security services, which are not described in this master's thesis.

2.2.3 Web Services

Web service is an interface, which describes a set of operations, which are accessible through a standardized XML-messaging via a computer network (Kreger 2001, 6. Gottschalk et al 2002, 1). W3C workgroup (2002b) on the other hand has described in their glossary Web service as a software system identified by a URI. It has public interfaces and bindings, which are described using XML. Web service's definition can be discovered by other software systems, which may then interact with the Web service in a way described by its definition. The interaction takes place using XML based messages transported by internet protocols.

Web services are considered interesting because they use a variety of widely accepted platform independent standards. This makes interaction between different systems possible. Tightly coupled technologies like EJB, DCOM and CORBA cannot measure up to a similar interoperability as Web services. (Tsalgatidou & Pilioura 2002, 136)

Figure 10 illustrates Web service actors, objects and operations in terms of Service-Oriented Architecture, similarly as presented in Introduction. The architecture consists of three roles, which are service provider, service requestor and service registry. The service and service description are objects which define how operations are performed. The operations which, can be performed are publishing, finding and binding. Web services are described in a ***service description***, which is written in Web Service Description Language (WSDL). Service description contains information for interacting with the service, such as message format, transport protocol and location of the service. The service description is published and retrieved from a Service Broker/Registry using UDDI (Universal Description, Discovery and Integration). (Gottschalk 2002, 170-171)

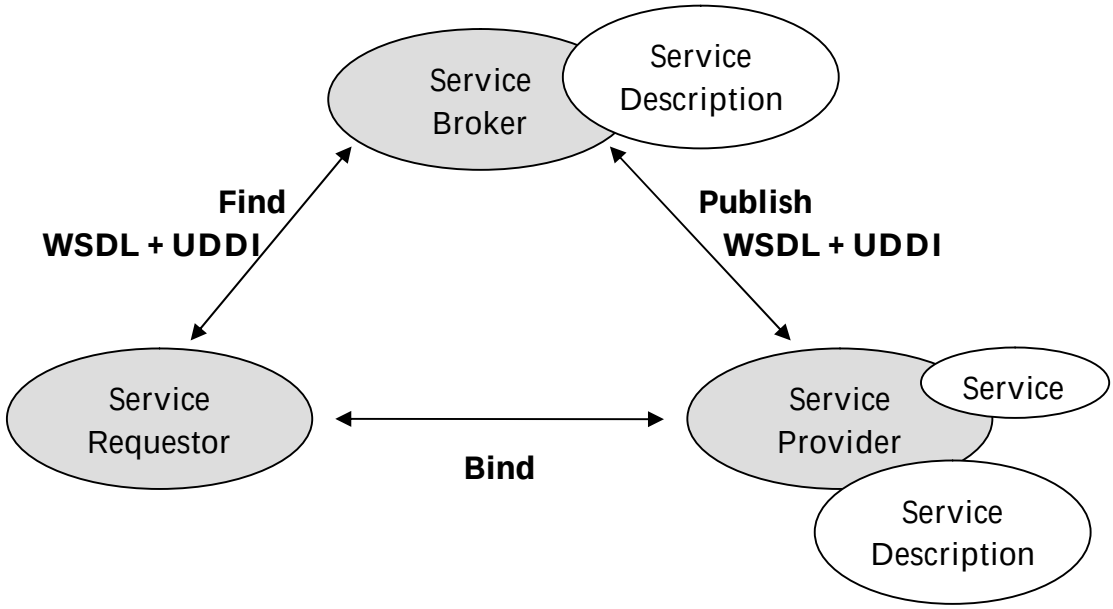


Figure 10. Web services roles operations and artefacts (Kreger 2001, 7, Gottschalk et al 2002, 171)

The publish, find and bind operations use SOAP protocol for communication (Gottschalk et al 2002, 171). ***SOAP***, known as Simple Object Access Protocol before version 1.2 (W3C 2000), is a lightweight protocol for the exchange of structured and typed information in a decentralized, distributed environment. SOAP uses XML technologies, which define an extensible messaging framework providing a message construct that can be transported over a variety of network protocols. The framework is independent of any programming model or language. (W3C 2003a. W3C 2003b)

SOAP consists of three parts: envelope, encoding rules and convention. Envelope describes, what is in the message. Encoding rules express instances of application-defined data types and conventions, which represent remote procedure calls and their responses. (Kreger 2000, 13)

WSDL is an XML format for describing network services. These are described as endpoints operating on messages and they contain either document-oriented

or procedure-oriented information. Both operations and messages are described in an abstract manner and they are tied to a concrete message format and network protocol to define an endpoint. WSDL can be used to describe any kinds of endpoints, regardless of what message formats or network protocols are used, though the specification only describes how it is used with SOAP 1.1, HTTP POST/GET and MIME. (W3C 2001)

Ariba, IBM and Microsoft have described the UDDI specification (uddi.org 2000), which describes how to publish and discover information about Web services. UDDI business registry, which is they key component of UDDI project, consists of three components: white pages, yellow pages and green pages. White pages include information on the business, such as address, contact information and known identifiers. Yellow pages contain industrial categorisation based on standard taxonomies and the green pages contain the technical information of the service, including a reference to the [WSDL] specification of the Web service. (uddi.org 2000, 2)

2.2.4 ebXML

EbXML (electronic business eXtended Markup Language) is a set of specifications to address several aspects of eBusiness in different types of businesses in an interoperable way (Gibb & Damodaran 2003, 138). EbXML-initiative is sponsored by United Nations Center for Trade Facilitation and Electronic Business (UN/CEFACT) and Organisation for Advancement of Structured Information Standards (OASIS). EbXML can be seen as an upgrade of EDI (Electronic Data Interchange) and its mission is to enable Business to Business commerce by using XML. (Gibb & Damodaran 2003, 4-5)

EbXML tries to address several issues EDI does not. As EDI focuses in the documents exchanged by trading partners ebXML tries to address also to transport, partner profile, business process and registry issues (see Figure 11).

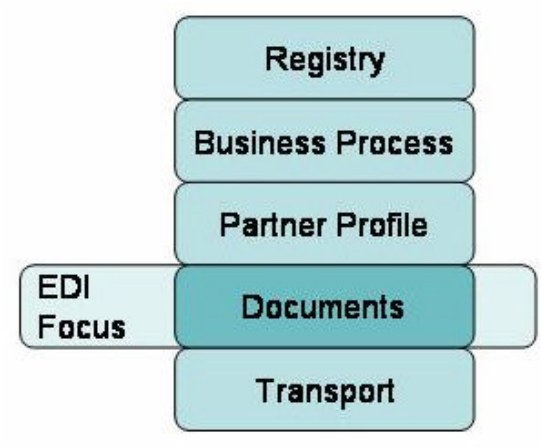


Figure 11. Scopes of ebXML and EDI. (adapted from Gibb & Damodaran 2003, 12)

EbXML's coordinating architecture consists of Registry, Business Process, Partner Profile, Documents and Transport –layers. Registry addresses need to have a standard interfaces and information model for hosting trading partner information, core components and other artifacts. Core components are used to maximize the re-use of electronic commerce documents and specify how to present data elements and meaning in a syntax neutral way. Business Process layer has standards for modelling business processing and an XML Schema for expressing them. Partner Profile layer has standards for an XML Schema for representing information about trading partners and agreements between them. The documents layer presents models of how to assemble business documents that consist of standard basic data elements. Semantics base on existing XML vocabularies and UN/X12 (EDI) experiences. The transport layer hosts a standard, secure and reliable XML and Internet based data transport service. (Gibb & Damodaran 2003, 12-19)

Communication methods are specified in ebXML Message Service Specification (ebMS). It defines the structures of exchanged messages between two ebXML Message Service Handlers (MSH) (Gibb & Damodaran 2003, 281). ebXML message consists of a communication protocol envelope, which can be e.g. HTTP, SMTP or even SOAP (Gibb & Damodaran 2003, 287-292).

Tsalgatidou and Pilioura (2002) see ebXML as a complex solution of Web services, which tries to address every aspect of the electronic business transaction. They state that ebXML and Web services should be used complementing each other. ebXML registries e.g. contain more information about the company than UDDI registries, which only describe the service.

2.3 Discussion on the Architecture and Comparing the Technologies

This section summarises the differences of different architectures and technologies and tries to describe why Web services and Message Broker architecture are the chosen as a research area.

2.3.1 Role of the Architecture

Services can be created with several different technologies in Service-Oriented Architecture. To enable these different platforms and technologies to communicate at least adapters are required. Web services are often seen as Point-to-Point type of integration. However, if a message broker would be used together with Web services it would probably bring similar advantage as in traditional message-oriented integration. When using other methods than Web services to provide services to other systems, broker could be used to integrate different technologies. The broker could also transform messages and therefore enable different types of request and reply messages and support version handling. For example older client version could still use a newer server

version. Also a message broker would help implementing transactional integrity and work flow rules. A message broker would provide a single point of managing transformations and transaction rules; this would ease the management of the integration architecture.

2.3.2 Comparing Technologies

RPC, DCOM and even CORBA eventually require single vendor's implementation, though they do support different platforms. DCOM requires often Windows platform and CORBA requires single vendor's ORBs. Though different vendor's ORBs do communicate together, they do not share the same security and transaction management methods. (Gisolfi 2001, 4) Web services on the other hand provide total interoperability and other capabilities crossing firewalls easily when using HTTP.

Wangler and Paheerathan (2000) discuss about the use of EJBs and CORBA in inter-organisational integration and they state that the distinction between EAI application servers and inter-organisational integration application servers is diminishing, and the need of Internet Application Integration (IAI) is emerging.

However, it would seem that at least in many cases Web services will be replacing the need of such application servers. Gokhale et al (2002) have compared Web services with CORBA. They state that everything that can be accomplished with CORBA can also be accomplished with Web services and vice versa, though the amount of effort required to implement the solution differs noticeably. As CORBA is a real object-oriented component framework, Web services are method of passing messages, with no support for objects. Gokhale et al (2002) state that Web services will some time sit on top of CORBA and sometimes CORBA sits on top of SOAP-like applications. CORBA and

SOAP are not exclusive, but rather they support and complement each other as they coexist.

Actually ebXML is a Web service just as SOAP/UDDI/WSDL, but here Web service is a term for services implemented with these technologies. Web services and ebXML are competing technologies, but they use a different approach. Web service initiative is using a bottom up approach when ebXML is using a top down approach. Web service initiative is implementing specifications that meet individual core requirements. EbXML can be seen as a complex implementation of the Web service model, which describes every aspect of eBusiness transaction. EbXML can currently use SOAP and HTTP for transporting the message, so ebXML can be used as a Web service payload. (Tsalgatidou 2002, 148-149) Web services are supported by major corporations like IBM and Microsoft and therefore they are likely to be the standard for eBusiness communication, though probably ebXML will have its users and it might be a popular message payload, since the message content is well defined.

In this master's thesis Web services are the key method of creating an architecture for Service-Oriented Integration. The following chapter describes different Web services architectures and standards.

3 WEB SERVICE ARCHITECTURES AND STANDARDS

This chapter deals with different Web service stacks. A Web service stack presents the components of a Web service architecture. It can be seen as a protocol stack, similarly as the OSI model from International Standardization Organization. The components of a Web service stack are different Web service standards that are used to describe Web services, to provide secure services and to orchestrate several Web services.

The IBM's conceptual Web service stack is one of the first published and therefore one of the most referred ones too. It is also a Web service stack that presents the role of Web service architecture and the standards in very through manner and therefore it is covered first in section 3.1. Together with IBM, Microsoft has been one of those organisations that have contributed most to the Web service standardisation by publishing several suggestions for standards. IBM's and Microsoft's latest view on Web service stack is presented in section 3.3. This chapter covers also W3C's Web Service Stack (in section 3.2), which can be seen as a vendor neutral stack. In some sense the IBM's conceptual Web service architecture describes the past and present of the Web services as the architecture presented by Microsoft and IBM shows the future of Web services, e.g. by replacing WSFL with BPEL4WS.

3.1 IBM's Conceptual Web Service Architecture

Kreger (2001, 10-32) defines IBM Web Service Architecture (see Figure 12) as a Web Services Stack, which consists of six layers: Network, XML-Based Messaging, Service Description, Service Publication, Service Discovery and Service Flow. In the figure Kreger has listed on the left side standards (e.g.

HTTP, SOAP, WSDL, UDDI and WSFL) that apply to the corresponding layer. The conceptual architecture also holds Security, Management and Quality of Service services.

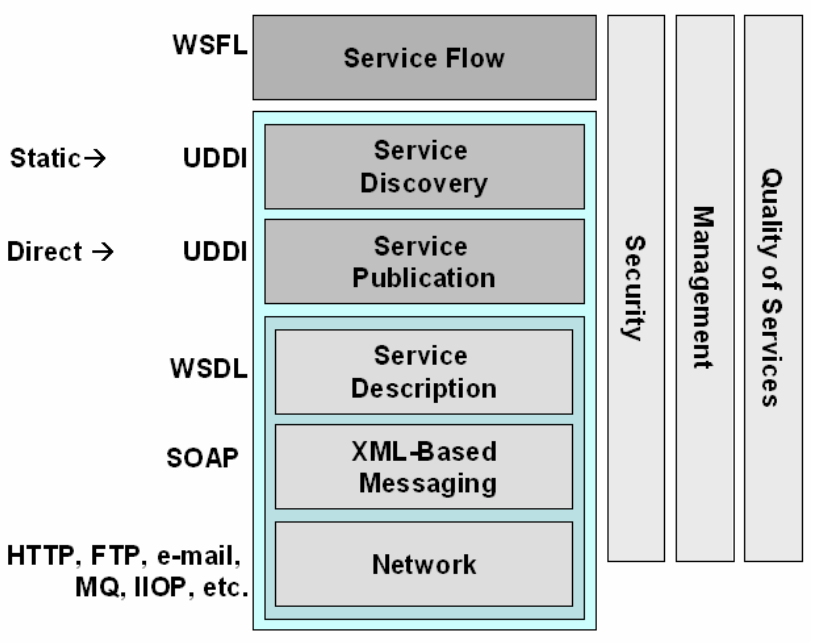


Figure 12. Web services conceptual stack (Kreger 2001, 10)

3.1.1 Interoperable Web Service Stack

The bottom layer, Network, makes Web services available in the network. Services, which are publicly available on the Internet use common Internet protocols, of which HTTP is a de facto standard, though SMTP and FTP are also supported. In Intranet messaging it is possible to use corporate messaging standards like message-oriented-middleware (e.g. MQ Series) and Remote Method Invocation such as CORBA IIOP. The XML-based messaging layer presents how SOAP is used as a message protocol. The Service Description Layer consists of a stack of WSDL description documents, which describe XML-based services. (Kreger 2001, 10-11) The usage of WSDL has been divided into two parts: service interface and service implementation. The ***Service Interface Definition*** is an abstract or a reusable definition – like IDL or abstract

programming interface – of a service, which can be instantiated and referenced by several Service Implementation Definitions. This allows for example definitions of industry standard services. The ***Service Implementation Definition*** is a WSDL document describing how a certain service interface is implemented. (Kreger 2001, 16) These three layers enable interoperable Web services and allow the implementation of low-cost Web services. The remaining layers are optional and are used for fulfilling business needs. (Gottschalk et al 2002, 171-172)

3.1.2 Publishing and Finding Web Service Descriptions

Services can be published either directly or dynamically. Direct publishing means that the service description is sent directly to the service requestor, e.g. by email after agreeing the terms of doing business on the Internet. Services can also be published in a more dynamic manner, where service descriptions can be retrieved from a given URL. The service's descriptions can be hosted in several different types of UDDI nodes, e.g. internal nodes, partner catalogues and electronic marketplace's UDDI nodes. (Kreger 2001, 19-20)

Similarly, acquiring of the Web service description varies in the same way as the publishing of the description. The requestor will find the Web service description either during the design time or during its runtime. The lookup services must support a query mechanism providing means to find a service for example by the type of interface (by using a WSDL template), binding information (i.e. protocols), properties (like Quality of Services), types of intermediaries required, taxonomy of the service and business information. (Kreger 2001, 21)

3.1.3 Service Flow

The Service Flow layer enables the composition of workflows from Web services and representation of simple Web services as one higher-level Web service hiding the other services. IBM suggests the usage of Web Services Flow Language (WSFL) as a language for defining service flows. (Gottschalk et al 2002, 174-176) IBM gave up WSFL specification and is now supporting BPEL4WS.

Leymann (2001, 6) has described WSDL as a language basing on XML for describing Web services compositions. It supports two types of service compositions: appropriate usage patterns and interaction patterns. The usage pattern of a collection of Web services describes how a particular business goal is achieved. This is typically a business process, and the created composition is often called orchestration, choreography or flow composition. The interaction pattern of the collection of Web services describes how the Web services interact with each other.

WSFL supports recursive composition meaning that each WSFL composition can itself become a new Web service. WSDL also supports both hierarchical and peer-to-peer interaction between partners. Hierarchical interactions are often found in stable long-term relationships, whereas peer-to-peer interactions are found in dynamically established operations on a per-instance basis. (Leymann 2001, 6 & Leymann et al 2002)

3.1.4 Security

Kreger (2001, 22) states that to fulfil the requirements of e-business, the Web services architecture must support security, reliable messaging, quality of service, and management of each layer of the Web services stack. Also service

context, conversations and activities, intermediaries, portal integration and service flow management should also be supported.

Web Service Security Layer must provide four basic levels of security: (a) ***Confidentiality*** guarantees that information is not made available or disclosed to unauthorized individuals, entities or processes and that the contents of the message are not disclosed to unauthorized individuals. (b) ***Authorisation*** grants access by checking the access rights and makes sure that the sender has the authority to send a message. (c) ***Data integrity*** checking makes sure that the data has not been altered or destroyed in unauthorized manner without noticing that the data has been interfered with. (d) ***Proof of Origin*** provides a way to identify the source of the message and also the possibility of proving that the message is not a replay of a previously transmitted message. This requirement informs of a possibility of error in data integrity.

Kreger (2001, 22-23) has discussed on the need of an additional security layer in Web services. She states that although the industry has provided methods for securing the transport layer, such as Secure Socket Layer (SSL) and Internet Protocol Security (IPSec) there are requirements for added security features. Also because of the dynamic nature of the Web services messaging the policy, trust and risk assessments must be revaluated. Service brokers, registries and meta-information providers have to be able to check who wants what information and if they have the right to access that information.

IPSec and SSL can only be used when using point-to-point communication. Because SOAP messages are processed by intermediaries, such as an integration broker, it is not possible to use IPSec and SSL, if there is not a trust among all the intermediaries. (Kreger 2001, 23) Using IPSec and SSL in non-trusted environments like over the Internet would not be possible.

Kreger (2001, 24) suggests that to be able to add cryptographic functions in new or existing applications, one should use middleware solutions, so that this functionality would be close to the application but not in the application itself. Intermediaries, such as an integration broker or a gateway between two networks, will forward the message further, and it has to transform security information like authenticity of the originator of the message to the next domain. Also, messages are stored on intermediaries when transported asynchronously, and data should also be secured from unauthorized access.

Kreger (2001, 24-25) presents a conceptual Web Security layer to address the security problems relating to the use Web services. This security layer responses to two different security issues: (1) Network Security and (2) XML Messages Security. To provide network security the security architecture must support SSL and HTTPS to provide confidentiality and integrity. The security of XML messages is addressed by three methods. (a) In case there are no intermediaries the sender can rely to the use of SSL or HTTPS. (b) The XML Digital signature (which is under the progress of standardisation by W3C) for verifying the originator of the message. XML Digital Signature defines a standard SOAP header and algorithms to produce a message digest and signing it with the sender's private key. (c) The architecture also supports trusted third party authentication service in the intranet. This service can for example be Kerberos.

To make it possible to secure XML messaging in end-to-end manner, one has to extend the security capabilities of flow and process systems. These should support multi-segment messages, which are protected with the public keys of the intended recipient. Kreger (2001, 23-24) has listed topics that need to be explored to fulfil these requirements: (i) It is the responsibility of the end-point to implement authentication and authorisation. These should be available for any exchange of information, defining which employees can use which services.

Intermediaries can perform authentication, authorisation and validation and verification of a digital signature, and they are also responsible of auditing and proof of origin. (ii) Service description layers should contain security oriented metadata to define access control and use digital signature, encryption, authentication and authorisation for the described Web service. (iii) Service requestors will use service description security elements to find suitable service end-points.

The SOAP envelope provides means to add meta-information to the message, such as transaction IDs and information on message routing and security. Thought the SOAP header contains the possibility of adding security functionality to it self, the SOAP specification does not specify such header elements. (Kreger 2001, 23)

3.1.5 Quality of Services and Reliable Messaging

In the IBM's Conceptual Web Services Architecture the Quality of Services provides relevant information for each the layer. For example, for the network layer it would enable the use different Quality of Service levels for different networks. (Kreger 2001, 25)

The network technologies will be chosen by their ability to fulfil the requirements of reliable messaging. Reliable messaging means "the ability of an infrastructure to deliver a message once and only once, to its intended target or to provide a definite event, possibly to the source, if the delivery cannot be accomplished."(Kreger 2001, 25)

According to Kreger (2001, 25) the network layer and the XML messaging together have to support four levels of quality of service for messaging: (1) ***Best-Effort***, is made so that the service requestor sends a message and the infrastructure and the service requestor will not try to retransmit the message.

(2) ***At-Least-Once***, means that the service requestor will make a request until it receives an acknowledgement, which can either be an acknowledgement of accepted request or an exception if the message cannot be processed. (3) ***At-Most-Once***, messaging extends At-Least-One, so that duplicate requests are not executed, for example using universal unique identifiers (UUIDs). (4) In ***Exactly-Once*** scenario the service requestor is guaranteed that the request has been executed. This is informed by a reply. Exactly-Once scenario eliminates the need of retransmission of request and accommodates failure scenarios.

The IBM Web Service Conceptual Architecture has a lot of components that are not required for an integration broker. For example if it is used in an internal network with message queuing systems which provide reliable messaging, one could use this message queuing functionality to provide reliable messaging, instead of using XML Messaging layer (Kreger 2001, 26). When using reliable messaging the applications and business process definitions do not have to be aware of, or manage intermediate states of message delivery (Kreger 2001, 26).

3.1.6 Systems and Applications Management

Services must be manageable on all the layers of the conceptual Web services stack and the Web services model components. One has to be able to manage (a) the infrastructure and (b) and the Web services. The management layer should provide reporting and recovery functionality of the network layer, XML messaging layer, service registries and Web service implementations. The partner organisation should have access to see status and health of the services they use. One should also report the performance, availability, events and usage metrics of Web services. (Kreger 2001, 27-28)

Service-Oriented Architecture and Web services can make usage of existing Work Flow middleware products, like MQ Series Workflow, if they support the

WSFL, HTTP and SOAP. The combination of company internal and external workflows is made with a gateway. The gateway can use HTTP and SOAP for the external public services and RMI over IIOP for intra-enterprise communication. (Kreger 2001, 38)

3.2 W3C Web Service Stack

W3C (2003c) has presented their Web service architecture, which describes functional components and their relationships to other components. The W3C architecture is a framework for future development of Web service standards. Figure 13 illustrates W3C's Web Service Stack. This stack differs a bit from the previous and more well known one (W3C 2002c), which divided the architecture to transport (the wire), description and discovery stacks. The W3C Web Service Architecture is divided into Communication, Messages, Descriptions and Processes layers and has also security and management features.

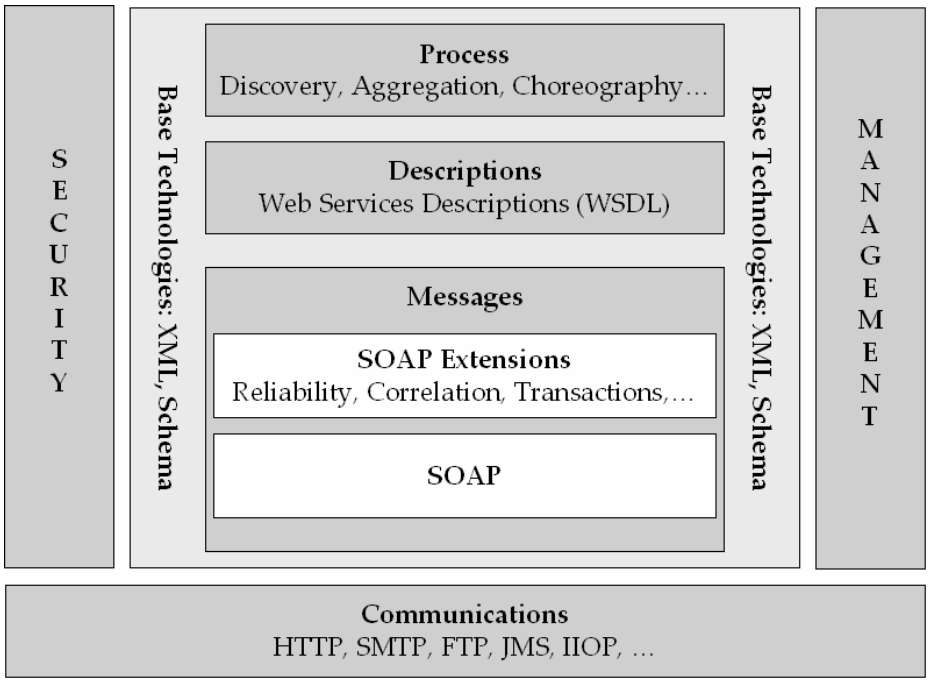


Figure 13. W3C's Web Service Stack (W3C 2003c)

3.2.1 Base Technologies, Communication and Messages

The whole W3C Web Services Architecture (WSA) is based on XML technologies, such as XML 1.x, XML Schema Description Language and the XML Base specification. XML enables the architecture to blur the difference of payload data and protocol data. Thus allowing mapping and bridging of messages over variety of communication protocols like HTTP, SMTP, FTP or application interfaces like JMS (Java Message Service) or IIOP. WSA requires that the communication layer exists and allows the messages to be tunnelled over other protocols, but it does not say anything about which protocol it uses and how the message is transmitted. (WC3 2003c)

On the other hand, according to W3C (2003c) the actual messages have to use SOAP and SOAP extensions. SOAP together with relating standards enable e.g. secure, reliable and multi-party messaging with authentication and encryption. The SOAP extensions are a part of envelope message structure (i.e. header and encoding rules) and can be used to provide additional information as routing and policy rules. As Web services become more popular the amount of built extensions will probably increase. (W3C 2002c)

3.2.2 Descriptions and Processes

The description layer contains information on how a service is invoked and how the service responds. The interface and implementation of the service are described in a WSDL document, similarly as in IBM's model. (W3C 2002c & W3C 2003c)

The Process Layer contains e.g. methods for service discovery, choreography of several independent Web services and aggregation of processes into higher level processes (W3C 2003c). As presented in IBM's Web Service Architecture the service publishing and discovery can be taken through in many different

ways. One of these methods is the use of an UDDI directory and an inspection language like WSIL (Web Services Inspection Language) (W3C 2002c). WSIL or WS-Inspection is a document that aggregates different types of service descriptions in one document. In other words, if a service has been described in several different languages the WS-Inspection document would contain a reference to all of these service descriptions. The standard also contains a set of conventions so that WSIL documents are easy to find. (Brittenham 2002) The standardisation project of W3C Web Services Architecture is unfinished and regarding the whole architecture and the layers presented above there are still several issues that are not covered.

3.2.3 Management and Security

The presented architecture contains also management functionality. W3C (2003c) define management as a set of capabilities for discovering the existence, availability, heath and usage of Web services and also the control and configuration of Web services, descriptions, agents of Web service architecture and roles undertaken in the architecture. The W3C architecture does not specify how Web services are managed, but it does identify key concepts relating to manageability. These are manageable elements (e.g. service description), its management capabilities, manageability interface and the manager.

The architecture also provides a secure environment for online processes. The W3C Web Service Architecture (W3C 2003c) describes a high-level abstraction of security considerations. The primary task is to ensure that intruders cannot access resources in which they do not have appropriate rights. The architecture should be reliably able to identify e.g. service providers and resources. The architecture should also ensure the data integrity of communications and transactions as well as ensure that information can only be accessed by intended parties. Also, entities which are not properly authorized should not be

able to access resources and actions, nor even know that the resource exists. Neither should anybody be able to prevent legitimate parties to access resources or perform actions (denial of service).

One should note that the architecture is not designed to combat against non-repudiation, mis-information and copy and replace. Also, the end user should be able to know how his/her personal information is used and the processed information should be handled with confidentiality and no unauthorized third party should be able to access it. (W3C 2003c)

To enable the use of security policies, there should be documents describing the policy and a policy guard for enforcing the policies. Policies can be divided into permissive policies, which actions and accesses are permitted to perform, and obligatory policies, which are actions and accesses which must be performed. The guard enables or disables action to a resource or an action. For example an intermediary is not allowed to forward a SOAP message if a security policy is violated. An audit guard monitors resources and agents and checks that obligations are fulfilled. The audit guard cannot prevent a security violation from happening but when it does it can then act on the process and some how try to remedy the situation.

3.3 IBM's and Microsoft's view on future Web Service Stack

Microsoft and IBM are partners developing Web service standards and have published together a significant amount of standard proposals. Microsoft (2003) has defined two Web service specification types: Baseline XML Web Services Specifications and Global XML Web Services Specifications. The baseline architecture consists of XML, UDDI, SOAP and WSDL, which provide a foundation for building basic Web services for application integration and

aggregation. The Global XML Web Services Specifications consist of such specifications as WS-Addressing, WS-Coordination, WS-Inspection, WS-Policy, WS-Referral, WS-ReliableMessaging, WS-Routing, WS-Security and WS-AtomicTransaction, which are described later in this section. These specifications, known as Composable Service Assurances, try to enable higher-level functionality such as security, reliable messaging, and transaction management (see Figure 14).

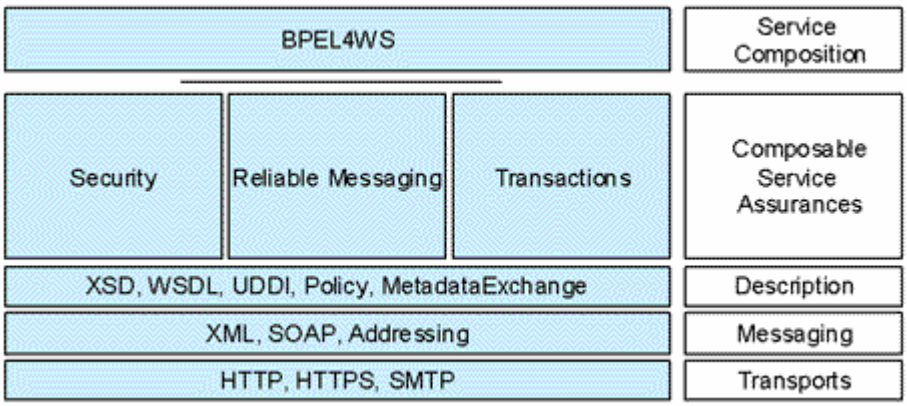


Figure 14. Interoperable Web services protocol architecture (Ferguson et al 2003)

The figure above describes Microsoft's and IBM's Web service stack. The three bottom layers are similar to IBM's and W3C's stacks' corresponding layers. Microsoft's architecture (which has been developed together with IBM) holds three groups of standards, called service assurances, which provide security, reliable messaging and transaction management. On top of all these is the service composition layer, which uses the Business Process Execution Language for Web Services (BPEL4WS) standard for orchestrating the co-operation of several Web services. (Ferguson et al 2003)

3.3.1 Baseline XML Web Services

The Message layer defines the supported message formats and in addition a transport interoperable method for identifying message senders and receivers with the WS-Addressing specification (Ferguson et al 2003). The specification describes an XML document, which normalizes underlying information into a format which is transport method independent. This helps Web services to be transmitted to other networks, which include intermediaries forwarding the messages, such as firewalls and gateways. (BEA, IBM & Microsoft 2003a)

The Web services are described with WSDL and XML Schema (XSD) and found in an UDDI-registry. XML Schema is used to describe XML data types and structures. The description layer also includes two new specifications WS-Policy and WS-MetadataExchange. (Ferguson et al 2003) BEA, IBM, Microsoft and SAP (2003a) describe WS-Policy as a model that extends other Web service specifications for presentation and communication of Web service policies. The WS-policy is a method for Web service application for specifying such policy information as authentication scheme, transport protocol selection, privacy policy and Quality of Service characteristics. The specification does not describe how these policies are implemented.

The WS-Policy specification contains also two other specifications WS-PolicyAssertions and WS-PolicyAttachment (Ferguson et al 2003). WS-PolicyAssertions is a specification used together with other Web services specifications to define common policy statements to achieve interoperability. The assertions are used within a policy and enable together with Web services and application-specific protocols the implementation of variety of policy exchange models. (BEA, IBM, Microsoft and SAP 2003b) WS-PolicyAttachment specifies how policy expressions are associated with WSDL type definitions and

UDDI entities. It also makes it possible to specify policies in a specific instance of Web service. (BEA, IBM, Microsoft and SAP 2003c)

WS-MetadataExchange specifies a method of providing information (metadata) about the service to clients through a Web service interface. This way potential user can retrieve information about the service and use it either during design time or runtime. (Ferguson et al 2003)

3.3.2 Security

The use of HTTPS and BASIC-Auth authentication do not work properly with intermediaries, therefore Microsoft and IBM together with their partners have suggested additions to security capabilities of Web services. The security family of the service assurance specifications contains the WS-Security, WS-Trust, WS-SecureConversation and WS-Federation specifications (Ferguson et al 2003). The WS-Security is the base of security functionality and provides message integrity and confidentiality. The specification also describes how to attach security tokens, a representation of security-related information (e.g. usernames, Kerberos tickets or X.509 certificates), to SOAP messages. WS-Security does not specify what kind of security tokens are required, its purpose is to be extensible and a general-purpose specification. (IBM & Microsoft 2002)

Message integrity is ensured with the usage of XML Signature and security tokens. The message confidentiality is ensured by enabling the encryption of parts of SOAP messages with XML Encryption. Both these mechanisms are designed to be extensible with other technologies and support operations of multiple actors. WS-Security also describes how one can use binary security tokens, especially how X.509 certificates and Kerberos tickets are encoded and how opaque encrypted keys are included to the messages. Again, the

specification is designed so that other security tokens can be added to messages and how these are described. (IBM & Microsoft 2002)

WS-Trust is an extension to WS-Security, which is used together with other Web service standards, and specifies how to request security tokens and manage trust relationships. Before accepting the service request the Web service can require that the request-message proves a set of claims, such as user name, key, permission, capability, etc. If such claims do not exist the service should ignore the message. The message is valid if it is associated with security tokens with the message and it proves with signatures that the requestor possesses those tokens. In case the requestor does not have the necessary tokens it can retrieve them from a security token service, which may require their own set of claims. (IBM, Microsoft, RSA Security & VeriSign 2002a)

WS-SecureConversation is another extension to WS-Security. It presents mechanisms to establish and share security context and specifies how to derive session keys from security context and how these keys are passed. The basic idea is that a security context token contains a shared secret, which can be used to create different keys, which both communicating parties can use to sign and encrypt messages. (IBM, Microsoft, RSA Security & VeriSign 2002b)

WS-Federation as well as WS-SecureConversation and WS-Trust are building blocks used to enhance security. WS-Federation specification describes how different security realms (e.g. different organisations) can provide a virtual domain, which supports authentication between Web services in different realms. Different realms are connected by security token services or identity providers, which have a trust between each other. Security token service (STS) is a Web service that issues security tokens, such as usernames and certificates. Identity provider (IP) provides authentication services to end requestors and data origin authentication to service providers, which is an extension to security

token service. STSs or IPs enable trusted communication between two realms. (IBM, Microsoft, RSA Security & VeriSign 2003)

3.3.3 Reliable Messaging

BEA, IBM, Microsoft and TIBCO (2003) have written WS-ReliableMessaging specification, which describes a protocol for reliable communication between distributed applications. It identifies tracks and manages reliable delivery of messages and also describes a SOAP binding to make the mechanisms interoperable. The specification allows it to be implemented in different network transport technologies.

The method of providing reliable messaging has been described in Figure 15. The initial sender sends the message to the Source, which handles the reliable messaging. The Source transmits and re-transmits it to the Destination until it receives an acknowledgement from the Destination, when it receives it. The Destination then delivers the message to the Ultimate Receiver. When sending several messages the messages can have sequence numbers and the last message is marked to be the last one. If a message from the middle is not received, the recipient will ask that message number to be resent. (BEA, IBM, Microsoft and TIBCO 2003)

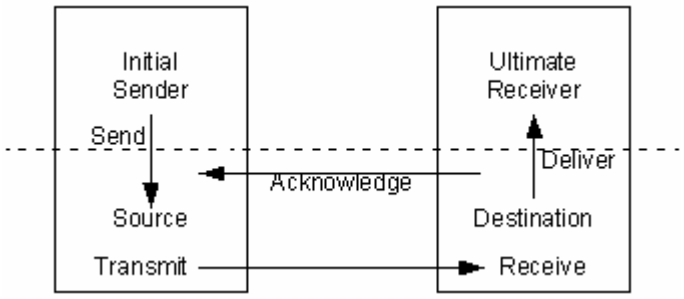


Figure 15. WS-ReliableMessaging, reliable messaging model (BEA, IBM, Microsoft and TIBCO 2003)

According to BEA, IBM, Microsoft and TIBCO (2003), messages can have four basic delivery assurances:

- At most once, every message is delivered at most once, without duplication or an error message is raised. Some messages may not be delivered.
- At least once, every message is delivered or an error is raised. Some messages can be delivered more than once.
- Exactly once, every message is delivered, without any duplications or an error is raised.
- In order, messages will be delivered in the same order they were sent. This delivery assurance can be combined with any of the three other delivery assurances described above.

3.3.4 Transactions

Web services perform different kind of activities that have a relationship. These relationships can be very complex and it can take a long time to perform them because of business latencies and user interactions. The WS-Coordination specification presents an extensible framework for the coordination of the activities by using a set of coordination protocols. WS-Coordination is a method which makes it possible to start, join and agree on an activity that consists of several tasks and messages. (BEA, IBM, Microsoft 2003b)

According to Ferguson et al (2003) the WS-Coordination has three elements:

1. Message element, which is called as a coordination context. The coordination context flows on all messages that Web services exchange while computing the activity. A coordination service uses the WS-

Addressing endpoint references in coordination context to identify the specific task being coordinated.

2. Coordination service, which hosts a service described with WSDL. This service is used to start and terminate a coordinated task, allow participants to register in the task and produce a coordination context, which is part of all messages in a group.

3. Coordination service, which provides also a WSDL defined interface for participating services. This service is used to inform the participants of the outcome of the coordinated task.

BEA, IBM, Microsoft (2003b) describe how the transaction coordination process works: Applications create a coordination context for an activity with the Activation service. The application acquires the coordination context and then sends it to another application. The coordination context contains essential information to register into the activity, which specifies the coordination behaviour that the application will follow. The application that receives the coordination context can use the Registration service of the original application or it can also use one that has been specified by an interposing, trusted coordinator. This way several Web services can coordinate their joint operations.

The Web Services Atomic Transaction (WS-AtomicTransaction) specification is written by BEA, IBM, Microsoft (2003b) and it is an extension to WS-Coordination. It defines how transactions can be coordinated in an atomic way by using a two phase commit. The model only takes in account the services involved, which can provide an interface for two phase commit but the actual implementation of the transaction management can use some other method.

The simple two-phase commit is suitable for long running internet computation. (Ferguson et al 2003)

WS-BusinessActivity is also a protocol set to extend WS-Coordination. It is used to build long-running, compensation-based transactions. BPEL4WS, which is described in section 3.3.5, defines transaction model for business processes, but WS-BusinessActivity describes the protocol rendering corresponding the business processes in BPEL4WS. (Ferguson et al 2003)

3.3.5 Business Process Execution Language for Web Services

BEA, IBM, Microsoft, SAP and Siebel Systems (2003) have published the Business Process Execution Language for Web Services (BPEL4WS) specification. The BPEL4WS specification describes a notation for specifying business process behaviour and Web service orchestration. The BPEL4WS (also known as BPEL) is complemented by WS-Coordination and WS-Transaction, which define specific protocols for the transaction processing and workflow systems (Papazoglou 2003, 51). The previous standards, Microsoft's XLANG and IBM's WSFL, are superseded by the BPEL4WS specification (BEA, IBM, Microsoft, SAP and Siebel Systems 2003). There are also several other competing business process modelling languages like WSCI, BPML, and BPSS (Wohed et al 2002, 2).

The BPEL4WS language supports the modelling of executable and abstract processes. An executable process describes the actual behaviour of the members of the interaction, i.e. it defines the execution order of different activities, which build up the process. It also specifies the exchanged messages, partners involved and exception handling. An abstract process is a business protocol, which specifies how messages are exchanged between different parties without

revealing the internal behaviour of the member parties. (BEA, IBM, Microsoft, SAP and Siebel Systems 2003. Wohed et al 2002, 3).

The actual processes are described in an XML document, where each processed element is called an activity. There are primitive activities, such as receive, invoke, reply and terminate and structured activities, such as while, pick and switch. The structured activities help defining the processing order and logic, whereas the primitive activities perform actions often related to messaging. (Wohed et al 2002, 3)

3.4 Discussion on Different Standards and Technologies

The Web services stacks presented in sections 3.1 – 3.3 have two different types of roles. The conceptual Web services stack from IBM and W3C's stack present the current situation of standardisation process. They also describe requirements for security and management, but do not really contain any solutions how to fulfil these requirements. The Web services stack from IBM and Microsoft on the other hand tries to address these requirements by providing a set of new specifications for security, reliable massaging and transactions. As mentioned in the beginning of this chapter the specifications presented in section 3.3 describe more were we might be going with Web services, rather than were we are. What Microsoft calls Baseline XML Web Services Specifications (SOAP, WSDL & UDDI) have already matured and received a stable position. The other specifications are still under hard development.

At the moment several different groups are producing Web services related specifications. These are often named as "initial public drafts" and "public specifications" and they demonstrate which parts of Web service specifications

require focus and standards. At the moment IBM and Microsoft have produced far more specifications on Web services than their rival group consisting mainly from Sun and Oracle. (Jennings 2003)

Fujitsu, Hitachi, NEC, Oracle, Sonic and Sun (2003) released their Web Services Reliability (WS-Reliability) specification, which competes with similar standards from IBM and Microsoft. WS-Reliability is a SOAP-based protocol for reliable messaging, including guaranteed delivery, duplicate elimination and maintenance of message order. In overall, the methods of accomplishing these goals are similar to the methods presented in WS-ReliableMessaging.

Also other sources have presented differing views, e.g. Sollazo et al (2002) have suggested modifications to IBM's Conceptual Web Service Architecture. They suggest the use of DAML-S for service descriptions and process composition. DAML-S in an extension to DARPA Agent Markup Language (DAML), and its purpose is to describe a computer-interpretable description of Web services (Ankolekar et al 2001, 1). It supports Web service description, automated Web service discovery, execution, composition and interoperation (Sollazo et al 2002, 2)

Jennings's (2003) view of how the standardisation process will proceed is a bit sceptical. He notes the problems Web Services Interoperability Organisation (WS-I) had in the production of Basic Profile 1.0, which included already quite well agreed SOAP, WSDL, UDDI, XML and XML Schema technologies. This was released by WS-I in August 2003, after about a years work (WS-I, 2003). Jennings (2003) predicts that a WS-Security 1.0 profile, including with test tools and PKI-based sample applications, will not be published until late 2004. This means, that the primary security mechanism in Web services will be SSL for quite a while.

This chapter described already some requirements for Web services architectures. The following chapters will present more requirements for integration and Web services architectures.

4 REQUIREMENTS FOUND IN LITERATURE

This chapter describes requirements for an integration broker supporting Service-Oriented Architecture. Some of the requirements are set generally for a Web services architecture, others are general requirements for an integration broker and issues organisations should keep in mind when moving to Service-Oriented Architecture. Though all presented requirements do not fit the purpose of this thesis directly, they are presented to give a holistic view of the requirements. In the end of this chapter there is a summary of the key requirements for an integration broker supporting Service-Oriented Architecture.

4.1 Reference Architecture for EAI

Puschmann and Alt (2001) describe reference architecture of Enterprise Application Integration System as follows. Its services are divided into three integration levels (see Figure 6):

Data integration level consists of ***connectivity services*** that take care of communication, addressing, delivery and security; and ***interface services***, that provide possibility to communicate through various application interfaces without changing the applications. This is done by providing ready built interfaces in order to ease programming and extend traditional Database Management Systems drivers.

Object integration level includes the core of EAI products: t***ransformation services***, which make object integration possible by receiving information, converting it to new format and passing it in the right format to recipient(s) through data

integration level. Transformation services require metadata, which describe elements of source and target application and transformation rules.

Process integration level contains process management services, runtime services and development services. ***Process management*** services provide similar functionality as transformation services, but are specialized in controlling workflow. Process management services collect messages and make numerous transformations to make sure that flow of information supports the defined process models.

Runtime services provide functionalities to support different EAI architectures, load balancing, scalability and also possibilities to monitor and analyse availability and performance of services.

Development services support building of new adapters, transformation models and specifying new process models.

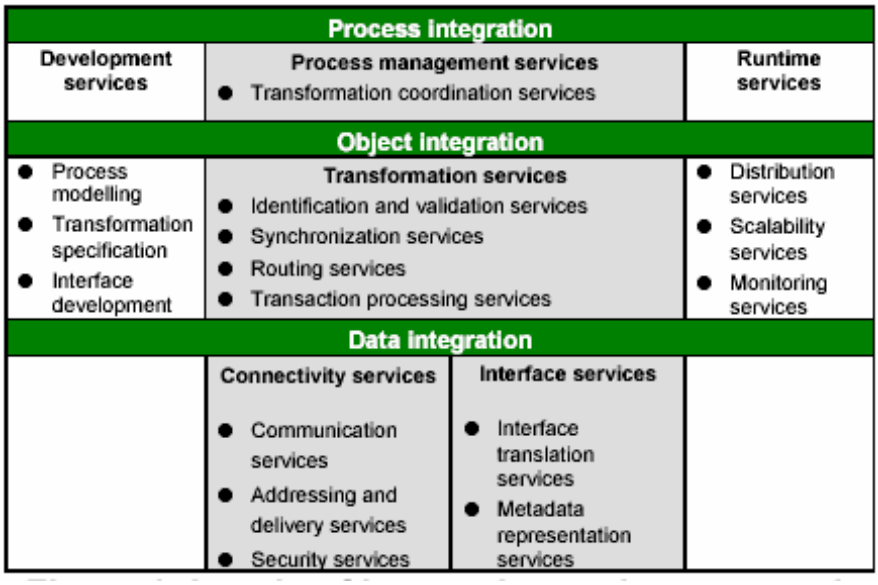


Figure 16. Integration levels (Puschmann and Alt 2001, 3)

4.2 Integration Broker and Web Services

Samtani and Sadhwani (2002b) have described both services of an integration broker and services that a broker should contain, if it supports Web services. According to them (p. 75-78) an integration broker should contain the following features:

- ***Enables all types of integration.*** The integration broker enables application-to-application (A2A), business-to-business (B2B), and business-to-consumers (B2C) integration.
- ***Interoperability***. The broker should be interoperable with existing applications regardless of the programming language and the platform they run on.
- ***Open Architecture***. The broker should provide an open, non-invasive and scalable architecture supporting major distributed computing architectures like COM+, CORBA and J2EE.
- ***Support for all communications protocols***. The integration broker should support all data transmission protocols used in the enterprise, including among others FTP, HTTP, HTTPS, SMTP and WAP.
- ***Directory Service***. The broker should contain directory services, which maintains the searchable index of all source and target applications, their locations and supported communication protocols and usage. It provides a single point of entry (i.e. gateway) for all the connected applications.
- ***Trading partner management and personalisation.*** The integration broker stores definitions, preferences and technical specifications (e.g. protocol and EDI/XML standard) for each application and this way makes it possible to personalize services to third parties.

- ***Security***. The broker provides security services like encryption, authentication, digital certificates and provides data privacy, data integrity, transaction non-repudiation.
- ***Scalability***. The integration broker should be scalable and fulfil user organisations requirements also in the future and enable load balancing and failover.
- ***Transactional integrity.*** The broker must maintain the transactional integrity, which means event-based processing, exception handling and recovery. In case a work fails all completed units of the work must be rolled back.

Samtani and Sadhwani (2002b, 78-79) claim that Web services will just be another service in an integration broker, and these services can also be provided with a third party solution. They have described the contents of an integration broker supporting Web services and service components: (a) The system should be able to receive, respond and generate SOAP messages. (b) In addition to SOAP it should support such Web services standards as UDDI, WSDL and XML. (c) The integration broker should be able to communicate in a secure manner, it should also provide secure connectivity to UDDI directories and other repositories. The broker should also support policy management and authentication and (d) provide transactional integrity. (e) The integration broker should provide mechanisms to audit access and usage of Web services and (f) monitoring tools to follow the health of the Web services networks. (g) It should also provide an environment for easy development, publishing, finding, and dynamic binding for Web services interfaces. (h) There should also be an easy way of connecting the broker with internal and external UDDI registries. (i) Also, the integration broker should support work flow management for using Web services in business processes.

4.3 W3C Working Groups requirements for Web Services Architecture

W3C working group (2002a) used two methods in gathering requirements for Web services Architecture: Critical Success Factor (CSF) Analysis and gathering of Usage Scenarios. The Critical Success Factor Analysis was used for top-down gathering of organisations' needs. The requirements were gathered to help development of Web services and are not directly applicable as requirements for a broker architecture.

W3C working group identified seven top level goals: (1) Interoperability, (2) reliability, (3) integration with World Wide Web, (4) security (5) scalability and extensibility, (6) team goals and (7) management provisioning.

Interoperability

Web Services Architecture (WSA) should be interoperable and support development of interoperable services regardless of the environment. The architecture should not assume any specific device or level of connectivity (e.g. wireless, intermittently connected) for clients or servers, or make any assumptions on the usability or visibility of services based on user location. The Web service architecture should be used from a spectrum of devices with different capabilities. The architecture should not be dependent of or preclude any particular programming model and it should consist of relationships between loosely-coupled components. The components should be well defined with unambiguous interfaces and their functional roles and responsibilities should be documented, including their inputs and outputs.

Reliability

The used architecture must be reliable and stable in the long run. It should also be adjustable and possible to be developed it while it is being used. The

architecture should be defined precisely, without ambiguity, by using standard definition languages when possible and by using standard and well defined new terms. The new version of the architecture should be compatible with the older versions.

Integration with the WWW

The Web services Architecture should integrate with the World Wide Web and be coherent with the current and future evolution of the Internet standards (WWW). The used architecture should not be in any unnecessary disproportions with the Semantic Web. WSA is consistent with current architectural principles of the WWW and it should enable device-independent Web services. The used architecture should adapt the internationalised character model for the WWW. Web service technologies used by the Web service architecture should be mapped to RDF/XML and the conceptual elements should be addressable via a URI. New architectural areas should be defined using XML Schema.

Security

The Web Services Architecture must be secure and provide an environment for secure online processes across distributed domains and platforms. When building the WSA possible threats should have been thoroughly analysed, such areas are (a) accessibility attacks (DOS, DNS, spoofing etc.), (b) authentication of the parties attending to the usage scenario, (c) persistent and transient authentication of authorship of data, (d) authorisation, (e) confidentiality, (f) data integrity, (g) non-repudiation of origin and receipt between transaction parties, (h) ways of expressing security policy, (i) means to access web service security policies, (j) auditing and (k) administrative functionality to administer the a-j features described.

The used architecture must protect the consumer of Web services across multiple domains and services. The Web Services Architecture must enable privacy policy statements to be indicated about Web services. Advertised Web services' privacy policies should be expressed in Privacy Preferences Project (P3P) and consumers must be able to access these policies. These privacy policies must be possible to be delegated and published. The architecture must also support interactions where one or more parties are anonymous.

Scalability and Extensibility

The Web Services Architecture must also be scalable and extensible. The architecture should provide modular components to provide appropriate functionality. In addition, the architecture should later be able to be developed and to fit the evolution of technology and business goals. The transport of data and data itself should be separated. Still the interaction scenarios should be simple. The System should not be precluded of quoting or modifying messages within other messages. One should also be able to develop and change components within the architecture independently of each other and still the system should work as intended.

The architecture should be kept simple and easy enough to learn, develop and maintain by its users. The architecture consists of conceptual integrity, rather than a set of disjoint ideas. The used architecture should also respond to requirements of switchover from traditional EDI to XML-based messaging and support peer-to-peer interaction of Web services. In order to provide means for transition from EDI the architecture must support reliable messaging and long running, stateful and choreographed interactions, both within and across trust boundaries. The Web Services Architecture should also support at least the following peer-to-peer messaging patterns: request-response, publish-subscribe and events and event notification. The Web Services Architecture must not

preclude persistent identities for peers, determining capabilities for peers or the usage of third party intermediaries (e.g. forwarding). It must enable direct peer-to-peer interactions without the use of third party intermediaries and also it must be possible to peers to discover other peers.

Web Service Architecture Team Goals

The WSA Working Group also states that the reference architecture the working group is gathering requirements for must meet the needs of the user community. They emphasize the role of architecture being reliable, stable and able to evolve over time and being consistent and coherent.

Management provisioning

The Web Services Architecture should also be a manageable and accountable environment for the Web services operations. The architecture must make it possible to manage and provision Web services. The used architecture should at least provide the following functions for system management: (a) Resource Accounting, (b) Usage Auditing and Tracking, (c) Performance Monitoring, (d) Availability, (e) Configuration, (f) Control, (g) Security Auditing and Administration, and (h) Service Level Agreements.

4.4 Integration Services Architecture

Gold-Bernstein (2003) presents an Integration Service Architecture (Figure 17), which presents the services that a service-oriented integration broker should contain. The Integration Service Architecture consists of User Interface Services, External Integration (B2B) Services, Data Integration Services, Translation/Transformation Service, Application Connectivity Services,

Integration Routing/Broker Service, Process Management, Workflow, and Application Orchestration Services and Security and Management [capabilities].

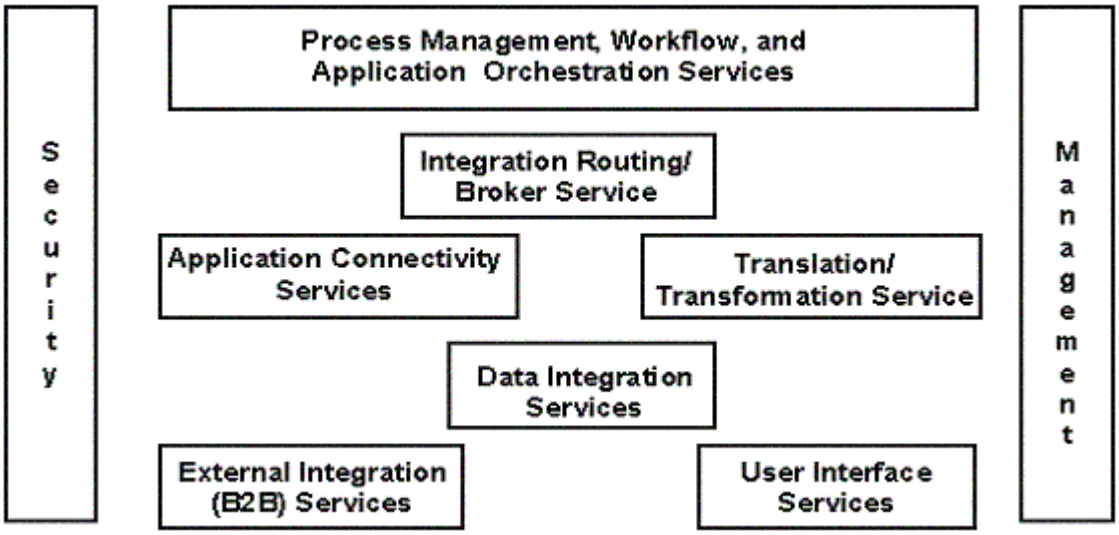


Figure 17. Integration Service Architecture (Gold-Bernstein 2003)

User Interface Services include technologies for presenting the information in different forms depending on the situation. For example, if an error occurs the alert can be sent either to the management application on a desktop computer or to a mobile device, depending on the time of the day.

External Integration (B2B) Services are used to manage connectivity options (such as XML and EDI) to different business partners. The B2B services should also contain process management, application orchestration, and workflow services.

Data Integration Services provide technologies for batch data transfer, database gateways, ODBC, JDBC and other technologies that provide means for data-level integration. These services contain semantic meaning of the data, which is captured during the transformation routines.

Application Connectivity Services contain adapters and Web service interfaces and protocols. More simple interactions can be done by using point-to-point architecture with suitable adapters, but when the amount of interactions increases it is recommended to prefer reusable adapters and connections to message brokers.

Translation/Transformation Services contain (a) transformation engine(s) and tools for mapping data and defining data translations. The definition process data requires the companies to define and correlate the semantic meaning of information in and across applications.

Integration Routing/Broker Service are typically provided by integration broker or rules engine. These technologies can be used for more complex integration project. The routing rules should be separated from transformation rules, so that they can be then deployed in several different places (like in a server or in an end-point adapter).

Process Management, Workflow and Application Orchestration and Services provide capacities for event-based processing and real-time management in Service-Oriented Architecture. Process tools provide ways to create business process rules with standards like BPEL4WS, BPMI, ebXML (see sections 3.3.5 and 2.2.4 for more details.).

Gold-Bernstein (2003) suggests that Systems Management tools should also be integrated. These tools work in several different layers and are used to manage networks, middleware, applications and processes. She also states that Security should be integrated throughout the architecture, to make single sign-on possible.

4.5 Delphi Groups Recommendations

Delphi Group (2002, 2-10) has listed issues and requirements organisations have to take into account when evolving Web services initiatives in organisations. Such issues are (1) development, (2) discovery, (3) security, (4) orchestration, (5) operations management and (6) Web Services Networks. The first five issues describe mainly requirements for Web services, though many of them suit also as requirements for an integration broker. Web Service Networks are an infrastructure in which Web services are deployed and provide infrastructure services, which the integration broker also has to provide.

From the development of point of view, the organisation has to select a platform on top of which it will develop its Web services. Development tools must also be available for that environment. One could generalize these Delphi Group's requirements and state that there should be a third party development community to develop Web services and adapters for the used architecture. Very few organisations develop their services in-house, which means that there must be a third party developer which can create these services.

Web services enable dynamic discovery. To make this possible clients and servers [also the integration broker] must have common definitions and concepts involving the Web services. There are standards like USML (UDDI Search and Description Language) and Web Service Inspection language (WS-Inspection) to make it possible to discover Web services.

The used architecture must be secure. The sender and the receiver of the message must be authenticated and valid content of the message and the possibility of tempered messages must be eliminated. Confidentiality secures that the information meant for a particular party is not seen by any other unauthorized parties.

Orchestration is required for defining and executing several synchronous and asynchronous Web services as a multi-step business process. In other words the architecture must support definition of business processes and work flow. These processes can require transactional services, which often are described to require the fulfilment of the ACID requirements (atomicity, correctness, isolation and durability).

Web services must be reliable and manageable. Messages sent back and forth during the Web service's life cycle should be delivered to the right recipient, in a correct amount of time, in the right order and in the correct number of iterations. In case of any faults there must be a method to alert both the sender and the recipient. Important aspects of managing Web services are a guaranteed delivery, non-repudiation and 'once-and-once-only delivery'. System should contain possibility of tracking the messages, especially, when the messages are asynchronous. Also real time information and logging of state of the Web services and information on the performance of them is important. The system should also support the possibility to set users different rights of access to various Web services.

Web Services Networks (WSN) provide infrastructure and such services as non-repudiation of messages, guaranteed delivery, 'once-and-once-only' delivery of messages, and authentication. WSNs provides also registry of categorized Web services that are available and meet the requirements of the requestors and providers. The registry makes it possible to search and find Web services. Web Services Networks task is also to manage application-to-application interactions.

4.6 Requirements from Web Services Networks

Truelove (2001) divides requirements of Web Services Networks in two broad categories: (a) Communication requirements, which are required for integration across firewalls over the Internet, and (b) Collaboration requirements, which are required to safely publish and consume or integrate Web services with collaboration of inter-enterprise processes.

Communication requirements consist of asynchronous, encryption, reliability and non-repudiation and polling services. Polling services mean that recipient does not continuously listening for incoming messages; but instead it periodically polls the sender for messages.

Collaboration requirements consist of access control, implementation abstraction, routing and registry & discovery. Implementation abstraction means that by creating an additional interface one can change and upgrade the Web services without breaking the systems that may be dependent on them. This is what is gained with the usage of an integration broker in SOA architecture. The task of routing services task is to orchestrate multiple Web services to support business processes and provide visibility into processes and manage the state of the processes. The Registry and discovery services are needed for publishing and finding the location. It is also used for binding and controlling access. This information is often stored according the UDDI specification. Registry and discovery services can either be public (Internet usage) or private (intra-enterprise).

4.7 Requirements for Adapters

As described in section 2.1.4 adapters are layers in-between the source/target application's interfaces and the message broker interface. This section covers requirements for the adapters. Beveridge (2000, 2-3) has described requirements for an adapter for legacy systems, these are: performance, integrity, scalability, exception handling and extensibility. The adapter must not constraint the execution, it mush have enough performance and be totally serialized. The adapter must perform correctly and it must avoid contentions. When the use increases the adapter should scale smoothly, determniscally and predictably. Exceptions, for what ever reasons they happens, must be handled. The adapter should also be easily extended to provide new services, but this extensibility must not compromise any other requirements.

Arsanjani et al (2003) have discussed business enterprise components, including adapters. They state that the components must be able to configure them self automatically, be able to alter themselves quickly and have characteristics of self description. The adapters have the same requirements as other Web services and applications the in infrastructure, they must be available, scalable, efficient and reliable. Adapters must also support dynamic monitoring for metering performance, manageability, auditability, replication and reliability.

An adapter making Web service requests does not have to be similar to one serving a Web service. A client adapter performs bit like a web browser, it posts a request and loads the SOAP replies. It of course also communicates with the integrated system: receives requests and returns responses through an API. As described in section 2.1.4 a Web service adapter (WSA) requires a platform to run on. In Kuebler's and Eibach's (2000) example it was a WebSphere Application Server containing the Web server. This running platform is here

after called WASP (Web Application Service Platform), which is used either to run the actual Web service or the adapter. A WASP contains a HTTP listener (Web server), which is needed to serve a Web service. To be able to request a Web service no Web server is needed. As we see, needs two types of adapter, ones for serving Web services and others for requesting them, though they can run on the same platform.

4.8 Discussion on found requirements

This section summarises the requirements presented earlier. This section presents a division for the requirements, which is developed from Puschmann and Alt's (2001) reference architecture for EAI and IBM's Conceptual Web Service Architecture (Kreger 2001). Since neither of these architectures notes adapters the presented division is modified to fit broker architecture. Therefore the requirements for the integration architecture are divided in (a) broker requirements and (b) adapter requirements.

Broker Requirements

In this thesis the broker's services are divided roughly in two different types (see Figure 18). The first services are those, which fulfil broker's main requirements. In other words, it has the functionality described in section 1.2, which are message transformation, rules engine and intelligent routing. These services are here after called Brokering Services. In this thesis Brokering Services consist of connectivity services, routing service, message transformation services and transaction services. Other functionality, security services, development services and runtime services, are called Supporting Services. The purpose of these services is to support the brokering, help managing the broker and develop e.g. rules.

Brokering Services	**Supporting Services**		
Transaction Services	Security Services	Development Services	Runtime Services
Message Transformation			
Routing Service			
Connectivity Services			

Figure 18. Requirements Classification for an Integration Broker

Brokering Services

Connectivity Services take care of communication, addressing and delivery. They should support a wide range of transportation protocols like: FTP, HTTP, HTTPS, POP, SMTP, and WAP. The brokering services can be seen as a gateway into the integration architecture, it supports several different protocols and makes it possible for example for the mobile users to uses systems with WAP.

Routing Service maintains a searchable index (UDDI Registry) of all source and target applications and their locations and information of which protocols they support and how the services are initialised. The routing service is responsible of routing the message to the right target application(s) and that the messages are delivered ‘once-and-once-only’.

Message Transformation Service makes it possible to transform the structure of the message, without changing the content. Also it can convert the content to match the target application, e.g. converting product IDs. The changes are mapped with metadata and processed by a transformation engine.

Transaction Services provide services for supporting business processes and workflow. It keeps the transactions synchronized and in case of errors it rolls back processed parts of the transaction.

Supporting Services

Security services enable encryption, authentication, authorisation, confidentiality, data integrity, and support digital certificates, and provide data privacy and non-repudiation of origin. It also makes it possible to set policies, user restrictions for both usage and search services.

Development services help building and managing transformation models and process models. Also it provides an environment for development, publishing, finding, and dynamic binding for Web services interfaces

The runtime services provide support for load balancing, reliability, scalability, extensibility, and management services. It enables monitoring of availability, configuration, performance, service level agreements and audits the usage of Web services.

Adapter requirements

Adapters make it possible to connect those applications that do not support the protocols used by the integration broker to be connected to the integration architecture. Adapter requirements are divided similarly to Adapter Services and Supporting services (see Figure 19). Adapter services consist of Connectivity Services and Interface Services. Supporting Services consist of Runtime Services, Development Services and Security Services. The requirements listed here are requirements of the integration architecture that also has to be fulfilled by the adapter, not just the broker. Also the Connectivity

services take care or the communication with different interfaces, instead of the broker doing that.

<table>
<tr><th>Adapter Services</th><th colspan="3">Supporting Services</th></tr>
<tr><td>Interface Services</td><td rowspan="2">Security Services</td><td rowspan="2">Development Services</td><td rowspan="2">Runtime Services</td></tr>
<tr><td>Connectivity Services</td></tr>
</table>

Figure 19. Requirements Classification for an Adapter

Adapter Services

Connectivity Services take care of communication, addressing and delivery between the adapter and the broker. Connectivity Services uses Web services to communicate with the broker.

The Interface Services are used to communicate with the back-end systems. They provide a way to communicate through a range of application interfaces like C, CORBA, COM+, J2EE and SOAP. It also makes it possible to connect to other middleware and databases, either by using proprietary database drivers or ODBC and JDBC. Interface services also support Web service standards like UDDI, WSDL and XML.

Supporting Services

Supporting Services consist of Management Services, Development Services and Security Services. The purpose of Supporting Services is to help the management and development of interfaces and provide security and efficiency to the adapter.

Runtime Services help monitoring the Web services and their state and following how the security measures are fulfilled. The adapter should also be reliable, scalable and stable. The adapter should work together with the broker and provide reliable messaging.

Development services are used to make changes to the adapters and to their interfaces. Development Services should make reuse of adapters possible.

Security Services provide encryption, authentication, authorisation, confidentiality, non-repudiation of origin, data privacy, data integrity, and support for digital certificates.

Chapter 5 uses the same division as presented above for presenting the requirements gathered for the integration architecture from the case company.

5 REQUIREMENTS BASED ON THE CASE-STUDY AND A SYNTHESIS

This chapter describes the requirements for an integration broker in Service-Oriented Architecture found in the interviews. The case-study and the case-study company are covered in section 5.1. Requirements, which are set to the broker, are presented in the second section. The following section describes the adapters' requirements. The latter part of this chapter summarises both the requirements found in the literature and the requirements presented in this chapter. The final section of this chapter discusses these requirements.

5.1 Describing the case-study

The requirements presented in the previous chapter were not purely requirements for an integration broker in Service-Oriented Architecture. Therefore requirements were also gathered from a case-study company. The requirements from the case-study company verify that the requirements from the literature were valid. The purpose was also to try to find new requirements for an integration broker in SOA. The presumption was that through a case-study one would find business related requirements. As a result of the case-study one can state that the previously presented requirements were verified and some new requirements were found, especially regarding maintenance.

The case-study company was UPM-Kymmene Oyj, which was selected because it was the originator of this research topic. UPM-Kymmene was interested in findin out what kind of an infrastructure Service-Oriented Integration with an integration broker would require. UPM-Kymmene is a large corporation of 35,000 employees. It is the world's leading producer of printing papers. The

company has an existing MOM architecture, in which it uses MQ Series from IBM.

The case-study was conducted via a set of interviews. Eight people were interviewed. The interviewees were selected together with a contact person from UPM-Kymmene. All the interviewees had existing knowledge on SOA and were selected from different organizational units to give a holistic view on the requirements.

Interviewee 1 is a service manager, who is responsible for integration services. He works in the IT services organization. Interviewee 2 is responsible of electronic procurements. Interviewee 3 is a project manager working with supply chains and technical integration. Interviewee 4 is a development manager working mainly with eBusiness related issues. Interviewee 5 is a manager at special applications competence center. Interviewee 6 is a manager in interoperability, working mainly with current integration architectures. Interviewee 7 is a director, responsible for different Mill Execution Systems. And Interviewee 8 is a manager in interoperability. He was heavily involved in the development of future integration architectures.

Together with the UPM-Kymmene contact person it was decided that study would focus on UPM-Kymmene's supply chain management. The interviewees were asked to prepare four use cases where Service-Oriented Integration could be used. The prepared use cases were asked to be more complex than simple request-reply -type, in order to cover the transaction and orchestration requirements. The interviewees were also asked to think in advance about (1) interoperability, (2) reliability, (3) security, scalability and extensibility, (4) integration with the WWW, (5) requirements from the users, (6) manageability, (7) transactions and workflow and (8) development (mainly based on W3C working group 2002a and Delphi Group 2002) in these use cases. Later, when

analysing these requirements they were divided into communication and Information Systems' security, interoperability, managing, reliability and scalability, integration with the Internet (and WWW) and system development. This division was done, because the gathered requirements seemed to drop into these categories and helped the building of the framework. Before each interview all interviewees were contacted by phone to confirm that they had received and understood the given material. Also they were asked if they were prepared for the interview.

The interviews were focused interviews (Hirsijärvi & Hurme 2001, 47-48), where there was no strict questions to answer. Hence, there was a set of topics to be covered. These topics were based on previously gathered requirements from the literature, mainly W3C working group's Web services Architecture requirements (2002a). These topics were listed to make sure that everything essential would be discussed during the interviews. Despite the guide list the each covered use case defined the structure of the interview. The interviews lasted from 40 minutes to 2.5 hours.

Unfortunately only one of the interviewees prepared four use cases. In most of the interviews two use cases were. In Table 1 it is presented how different answers from each interviewee spread to each category. One can see from Table 1 and Table 2 that the amount of requirements varied very much between different interviewees. In total, the interviewees found 217 requirements, but one must note that this figure contains same requirements presented by different interviewees. The majority (35%) of these were communication and Information Systems' security requirements. Requirements regarding manageability, reliability and scalability were presented, 63 times, which is 29% of all requirements.

Table 1. Amount of requirements to each category from each interviewee

Requirement topic	Interviewee								
	1	2	3	4	5	6	7	8	Total
Communication and Information Systems' security	14	15	4	5	4	13	6	15	**76**
Interoperability	3	5	1	1	2	8	1	19	**40**
Managing, reliability, scalability,	13	5	4	6	3	11	0	21	**63**
Integration with the Internet (and WWW)	2	2	0	4	0	3	1	5	**17**
System development	6	1	0	0	6	2	0	6	**21**
Total	**38**	**28**	**9**	**16**	**15**	**37**	**8**	**66**	**217**

Table 2. Percentages of the amount of answers given by an interviewee regarding each requirement topic and in total

Requirement topic	Interviewee								
	1	2	3	4	5	6	7	8	Of total
Communication and Information Systems' security	18 %	20 %	5 %	7 %	5 %	17 %	8 %	20 %	**35%**
Interoperability	8 %	13 %	3 %	3 %	5 %	20 %	3 %	48 %	**18%**
Managing, reliability, scalability,	21 %	8 %	6 %	10 %	5 %	17 %	0 %	33 %	**29%**
Integration with the Internet (and WWW)	12 %	12 %	0 %	24 %	0 %	18 %	6 %	29 %	**8%**
System development	29 %	5 %	0 %	0 %	29 %	10 %	0 %	29 %	**10%**
Of total	**18%**	**13%**	**4%**	**7%**	**7%**	**17%**	**4%**	**30%**	**100%**

On the other hand there were very few requirements for integration with the Web (8%) and system development (10%). Interoperability was covered quite well with 40 requirements. A more alarming issue regarding this research is the significant variation between the amounts of presented requirements between different interviewees. Interviewee 8 presented in total 30% of all requirements gathered, interviewees 1, 6 and 2 also gave a noteworthy amount of requirements. One can see the difference between interviewees that had been

previously contacted with MOM and SOA and had thought in more detail of the benefits of Service-Oriented Integration. These interviewees (1, 2, 6 and 8) gave the majority (78%) of all requirements. On the other hand, the interviewees 4 and 5 presented a notable amount of requirements in their own interest area (integration with the Internet and system development).

In the case-company there were few experts in the research area. This explains the imbalanced division of requirements between different interviewees. To receive enough interviews one had to find people who had some knowledge in SOI, but had not thought the use of an integration broker with SOI. Also when preparing the case-study the presumption was that persons from different parts of the case study organisation would have different views of SOI requirements, especially regarding business requirements. This happened in some level, but mainly the level in which use cases were discussed was so technical that business did not raise any unexpected or important amount of requirements. Or the interviewees did not have a specific opinion on how things should be implemented.

Despite the uneven division of requirements the case study did raise requirements not presented in literature. These requirements were mainly for manageability and the use of Service-Oriented Integration with a message broker.

5.2 Requirements for the Broker

The main roles of an integration broker (presented in the introduction) remain the same in SOA and Message Oriented Middleware (MOM) environment, they are message transformation (interviewee 6, interviewee 7, interviewee 8 2003), rules engine [including workflow] (interviewee 1, interviewee 6, interviewee 8

2003) and intelligent routing (interviewee 2 2003). This section describes requirements for the broker. These requirements are divided in Connectivity Services, Routing Services, Message Transformation, Transaction Services, Security Services and Development Services, as presented in section 4.7.

5.2.1 Connectivity Services

Ideally, organisation's current Message Oriented Middleware infrastructure should be used for reliable communication, e.g. MQ-networks. If HTTP is used, it should be run over the current messaging network. (interviewee 6, interviewee 8 2003) In addition of using the existing messaging networks, the broker should support both existing in-house and global standards. (interviewee 6, interviewee 1 2003)

Instead of using HTTP, one should use HTTPR or a protocol fulfilling the same requirements, especially when communicating over the Internet (interviewee 1, interviewee 8 2003). HTTPR guarantees that in of case errors, the sender knows that the message was not transmitted to the recipient.

The architecture used should not prevent the usage of any technology (including mobile devices). The architecture should consist of a broker and several gateways, which give access to proprietary software, mobile devices (e.g. SMS, WAP), Internet-access and applications with restricted Web service capabilities. (interviewee 6, interviewee 7, interviewee 8, interviewee 2 2003) The broker should be able to transform messages from one protocol to another (interviewee 6, interviewee 8 2003). For example, if an organisation is running Web services over MQ-network in the Intranet, they must change the transport protocol (e.g. to HTTPR) when sending a message to a partner over the Internet.

The system should be able to use also other ports than port 80. Such functionality could be needed for example if a virus would exist in the

corporate network and port 80 should be closed because of the virus. In this case, if the architecture would use some other port, the Web services which communicate with partners could still be used. (Interviewee 6 2003)

Interviewee 5 stated, that probably indisputableness is not an important requirement for UPM-Kymmene, this means that in case of an argument of whether the message was transmitted or not the organisation does not have to be able to confirm that the message was received by the receiving organisation.

The integration architecture should support both synchronous (interviewee 1, interviewee 2, interviewee 3, interviewee 5, interviewee 7 2003) and asynchronous (interviewee 2, interviewee 5, interviewee 3, interviewee 7 2003) messaging. Synchronous services are typically the ones used by people or applications which need an instant response.

It is very likely, that a third party instance will be used when communicating with partners. Such third party instances are hubs and electronical marketplaces and auctions. The role of the hubs is to contain information (such as product catalogues, routing information) of different companies and route messages (request and replies) to the right recipient. Also, if needed, the hub will transform the message to fit the recipients systems. (interviewee 2 2003) When communicating with partners, the communication methods should be standard to make the interaction as easy as possible (interviewee 6 2003). One should be able to provide partners a possibility to test the availability of the services (i.e. like a ping), this service would be available for all partners (interviewee 8 2003).

5.2.2 Routing Service

The routing service routes the messages to the correct recipient (among others, interviewee 3 2003). These messages must be sent only once and received only

once (interviewee 3, interviewee 5, interviewee 7 2003). In case a message cannot be delivered, an error message must be sent to the sender. (interviewee 3, interviewee 8 2003)

The broker architecture should contain its own service registry, which describes the services. When an organisation uses services provided by some third party, these could be published in an internal service registry, so that developers know these are safe to use and firewall and network settings are configured correctly. (interviewee 1, interviewee 5, interviewee 6 2003)

The registry must be able to contain information about different versions of the same service. The broker would be used to enable older client versions to use newer version of the Web service. The broker would transform the request to fit the newer server version or route it to a instance of the older service. (interviewee 7, interviewee 8 2003)

In some cases the same service could be available at several locations. The broker can try to connect to services in an order which has been defined in setups. Or, the broker can try to find an available service, by polling them and connecting to the first one that responds. (interviewee 8 2003)

A message can have strict requirements of delivery time. Such messages are e.g. when processing SAP's master data, customers and their information. Also, when a human user is using services, the systems should serve the user in real-time. (interviewee 5, interviewee 8 2003)

5.2.3 Transaction Services

An open issue is how much logic the broker should contain. As the amount of logic increases, the complexity of the system increases also. One wants to avoid situations in which the same logic is placed in several different places. The

interviewees 4 and 6 would rather have an integration architecture, which would not contain so much intelligence (like running rollbacks, processing several queries with one request and combining messages) and complexity.

UPM-Kymmene's business does not hold long transactions. There for the requirements for Transaction Services might be different in another kind of business. Broker's role should more be assisting than actually managing the transactions, which should be described with a workflow description language. This would mean that the actual services would have ***commit*** and ***rollback*** procedures to provide transactional capabilities. Should a broker notice an error, it would inform the requesting service about it, and the service would perform necessary rollbacks locally and remotely. Broker must take care of the order of messages, so that they arrive in the order defined by the administrators (interviewee 1, interviewee 3, interviewee 4, interviewee 6, interviewee 7, interviewee 8 2003).

Interviewee 6 speculated that if the transaction and brokering (routing) engines could run consecutively in different machines, this might reduce the amount of complexity.

5.2.4 Message Transformation

The broker must be able to transform the content of the messages (interviewee 1, interviewee 5, interviewee 7 2003). For example, as different systems might have different product or customer numbers, the broker must transform these to fit the message recipient. As presented in the previous section the system must be able to convert the differences between the previous and the current versions of Web services (interviewee 7, interviewee 8 2003).

In some cases the same service could be used from several locations. This would be done to ensure availability. Still, despite the location of the service

and the format of the service request, it would be preferred that there would be only one generic service request for that service and the broker would transform that request to fit the right recipient. (interviewee 8 2003)

5.2.5 Security services

In general, the broker must fulfil all the typical security requirements, such as: authentication, confidentiality, data integrity and non-repudiation of origin (interviewee 5 2003). One of the main requirements, which was often presented by the interviewees, was authentication of the service provider. When using a service, the client (requestor) must be sure, that the service provider is who it states to be (interviewee 1, interviewee 2, interview 4, interview 5, interviewee 7, interviewee 8 2003). This way, the client knows that it can safely use the service and trust the response it gets. In addition to the broker letting through only permitted requests, the broker might also have to hold priorities, which service request and replies should be transmitted first (interviewee 8 2003).

The service does not always have to authenticate the client. In some cases the services are totally open to free use for all users. Such service could for example be a service providing currency exchange rates. (interviewee 1, interviewee 8 2003) But on the other hand, some services have to authenticate the user to be able to serve the user correctly (interviewee 2, interviewee 5, interviewee 8 2003). Such situations can for example be, when users have different access rights to the data.

The broker needs to authenticate the clients and recognize where the message is bound. This way the service does not receive unnecessary service requests. By adding authentication methodology both to the integration broker and the actual service one can add another line of defence. (interviewee 2, interviewee 6, interviewee 8 2003)

One must also remember that when authenticating the client, the authentication process might consist of authentication of the client-application, or the computer running the application, or the person using this application. Also authentication can consist of authentication of all these three elements. (interviewee 6 2003) The authentication does not guarantee that the message content has not changed during transfer. There for there must exist methods to check that no-one has tampered the data in the message (interviewee 3, interviewee 8 2003). The architecture should authenticate all these elements, without the need of passing variables indicating the user, application and computer inside the message (interviewee 8 2003).

Depending on the situation, data is or is not required to be encrypted. For example law requires that Human Resources information to be encrypted. Encryption was also preferred when communicating over the internet (interviewee 1, interviewee 2, interviewee 2, interviewee 4, interviewee 5 2003).

In some cases it would be enough, that encryption and transport would be done with HTTPS (interviewee 5 2003). An issue dealing opinions was whether encryption should be done in the application level (interviewee 2 2003) or in the transportation level (interviewee 7 2003). The transportation level encryption might normally be enough, but when the data is stored in a database as encrypted, it should be the applications that encrypt the messages (interviewee 7 2003).

When communicating over the Internet the architecture should support authentication of the service by the use of digital certificates. The architecture should also use digital signatures for proving the message has not been tampered. Both digital certificate and signature base on Public Key Infrastructure (PKI), which can be used for both encryption and authentication. (interviewee 1, interviewee 3, interviewee 4, interviewee 6, interviewee 8 2003).

When communicating with third parties the information systems security relies mostly on the firewalls etc., therefore the broker does not have such an important role when securing systems against hackers. Still, there must be one point of access for the Web services and service directories, these can be called gateways. The services available for partners (e.g. communication through the internet or via leased line) must be restricted and they can only see and access services which they have right to use (interviewee 1, interviewee 2, interviewee 4 2003).

It would be very unlikely, that UPM-Kymmene would host any publicly available services. This means that only authenticated and authorized users have access to UPM-Kymmene's Web services. Also, the outbound access to public services would be restricted. This means that, any public services which are described in internal registries, would be tested and accepted and stated to be safe to use. (interviewee 5, interviewee 6, interviewee 8 2003)

5.2.6 Development services

The development environment is used to build new transformation and routing rules. Also service registries are updated and maintained with these tools. It would be preferred, if these tools would use Eclipse platform (interviewee 1 2003). Eclipse is an open platform for tool integration, provided by several tool providers like Borland, IBM, Rational Software, Red Hat, SuSE, Sybase, Fujitsu, Oracle, SAP, Ericsson and Intel (Eclipse 2003).

The development environment must support work of several developers. It must lock the items other developers are using (interviewee 1 2003). In other words, the system must have CVS (Concurrent Versions System) capabilities. Users should have a shared desktop, which would show all the running processes and their state (interviewee 1 2003). Developers, at local sites and

departments, developing new services should have access to the broker and the service registry, and they should see the state of their own services, which they are authorized to see (Interviewee 2, interviewee 4, interviewee 6 2003).

The service registry should be centrally managed with the development tools (interviewee 1 2003). When new services are published, they should be available to other developers as automatically as possible. This means, that there should be a centralized registry, instead of storing service descriptions at all clients. These service descriptions should work in an interoperable way with the development tools, so that developers can publish and retrieve service descriptions both from and to internal and public service registries (public registries as registries for certain branch of business). (interviewee 8 2003)

Also the development tools should support the maintenance of several version of the same service. Either the request is converted to the newer message format or routed to an instance of the older service. Similarly the replies would be converted to the older messaging format. (interviewee 7, interviewee 8 2003)

Third party developers should have access to the service descriptions. Also partners and other organisations that are using these services should be able to retrieve these service descriptions, so that they can build their own solutions. The services, service descriptions, security and other functionality should be easy to manage. When the amount of services increases, the system should maintain its manageability. (interviewee 5, interviewee 8 2003)

The architecture should support the use of physically separated development, test and production environments (interviewee 6 2003). Changes and settings of these environments should be documented in an unambiguous way, so that it supports maintenance. The tools provided should support such documentation. (interviewee 7 2003)

5.2.7 Runtime services

Runtime services provide load balancing, scalability and management capabilities to the integration broker. Though, the interviewees thought that message loads can be estimated and there for scalability and load balancing are not so important, there will be momentarily peaks in message traffics and scalability is necessary to handle these peaks. The architecture must have enough buffering capability, so that data will not be lost during error situations. (interviewee 1, interviewee 2, interviewee3, interviewee 4, interviewee 6, interviewee 8, 2003)

The system should alarm the administrator(s) when problems occur and in these situations the system should require minimum human interaction. The system should inform the administrator(s) if a connection to a service is lost. (interviewee 1, interviewee 6, interviewee 8 2003)

There must be monitoring tools for monitoring usability, settings, availability, messaging volumes, time, performance, target and sources. These tools can be used e.g. to find bottle-necks and to follow the availability and response times of partners' and third parties' services. Also one must be able to follow the status of the message. For example in case of an invitation of a bid, there could be such statuses as delivered, under processing, replied, etc. (interviewee 1, interviewee 2, interviewee 4, interviewee 5, interviewee 8 2003)

Two most important issues regarding runtime services are the reliability and usability of the broker. The broker can be seen as one single point of failure, if the broker is not available, the whole messaging infrastructure is down. The broker must be available all the time. Also speed and performance have value in some cases. (interviewee 2, interviewee 4, interviewee 5, interviewee 6, interviewee 8 2003)

5.3 Requirements for the Adapters

Adapter's role in the integration architecture is to work between the broker and the source/target application interface (Linthicum 2000a, 309). An adapter has to be able to fit different kind of systems to the same integration architecture (interviewee 6 2003).

When building an integration architecture with Web services the adapters can run on the same Web Application Service Platform (WASP) (interviewee 8 2003). These WASPs can contain a Web server, which is used to host be services when using HTTP for the interaction.

An adapter can either be used to service similar applications from different locations or one adapter can serve one physical location (campus model). The WASP should support both methods. The WASP should be simple, light and inexpensive, since it is distributed in several places. It must support remote management and monitoring and the management should happen with the tools provided by the integration infrastructure. (interviewee 8 2003)

This section describes the requirements found for the adapters. Requirements for connecting the adapter to the integration broker are described in section 5.3.1. The following interface section describes the requirements for interaction with the back end systems. Requirements for managing and developing the adapter are described in the Runtime services and Development services – chapter and the security requirements of the adapter are presented in the last section.

5.3.1 Connectivity Services

Connectivity services provide a Web service interface, through which the adapter communicates with the Web Service Network and the broker. This end

of the adapter must be totally interoperable. From the service point of view the architecture can use HTTP for communications, but rather it should use existing MOM, in UPM-Kymmene's case MQ-networks. The adapter should be able to transform the service's RPC or HTTP communication to MOM compatible communication. (interviewee 8 2003)

5.3.2 Interface Services

Adapters must provide such interfaces that current systems can use them. (interviewee 8 2003) There must be an adapter supporting the ERP solution (in UPM-Kymmene's case SAP). This adapter must support SAP's IDOC, RPCs and programming and interface language ABAP, which has a very service like ideology. (interviewee 1, interviewee 2 2003) A WASP should be able to run Unix's RPC and IPCs and support different component technologies like Microsoft's COM+ and DCOM [and Sun's EJB]. (interviewee 8 2003) UPM-Kymmene's Mill Execution Systems (MES) are built with COBOL, C/C++ and use SQL-databases. The adapter should be connected to these systems. (interviewee 4 2003) Some services can be behind a transaction engine like Tuxedo, in this case the Tuxedo should be integrated to this architecture and an adapter should provide the services available in Tuxedo environment (interviewee 6, interviewee 3 2003).

When communicating with databases, an adapter should be used. The broker should not have straight connections to the database. The adapter should communicate with the most common databases, support their native drivers and support ODBC and JDBC and it would act like a database front end. (interviewee 2, interviewee 6, interviewee 8 2003)

Proprietary systems, even if supporting Web services, might not support the used integration architecture. For example, these systems might not allow the

use of a broker, even less another transport method than HTTP. This means that with an adapter, one must fool the proprietary system to believe it is discussing with an UDDI registry and later straight with a Web service, even though it is just communicating with an adapter. This kind of capabilities could be needed e.g. when the message transportation is not using HTTP but the existing MOM. (interviewee 6 2003)

5.3.3 Runtime Services and Development Services

Adapters must be reliable. Probably adapters are not duplicated, so they must be 100% reliable. Or the architecture should be able to route requests to another adapter, if the primary one is unavailable. (interviewee 8 2003)

The adapter must be able to retrieve service descriptions and configure itself, so that it becomes a Web service client. When further developing a service, the adapter should be configurable with parameters, without changing the adapter's actual source code. The development tools must cooperate with the adapters, so that service descriptions and definitions can be updated at the adapter with the development tools. This should happen semi-automatically, e.g. the tools should support batch processing. The adapters should be configurable, and this configuration should be able to be done by using the tools belonging to the integration architecture. (interviewee 8 2003)

5.3.4 Security Services

The adapter must support the architecture's authentication methods, so that the backend system can either authenticate the server or the requestor (interviewee 8 2003). If XML's security measures are used the adapter should support them (interviewee 8 2003). If the application behind the adapter cannot encrypt the message content the adapter should do it. (interviewee 6 2003)

5.4 Synthesis of Case-study and Literature Requirements

This section gathers the requirements described previously in chapters three and four. The requirements are divided to broker's and adapter's requirements.

5.4.1 Broker's Requirements

As presented in 4.8 and 5.2 the broker's services are divided into Brokering Services and Supporting Services. Brokering services are further divided into Transaction Services, Message Transformation, Routing Services and Connectivity Services. The Supporting Services have been divided into Security Services, Development Services and Runtime Services. This section describes these services and the requirements for these services, both found in literature and during the interviews. One of the broker's main requirements is to ease transformation of one Web service version to another. The broker transforms the incompatible message to fit the recipient or it routes the message to an instance of the Web service supporting the older messaging format.

Connectivity Services

The role of Connectivity Services is to take care of communication, addressing and delivery of the messages. These services should support several protocols like: FTP, HTTP, HTPR, HTTPS, POP, SMTP, WAP and MQ. The connectivity services works as a gateway, by enabling systems using different communication protocols to join the Web Services Network. The architecture should use the existing MOM infrastructure to provide reliable messaging. HTTPR or a similar protocol could be used when communicating with partners. The internal messaging should still use MOM and transformation to HTTPR would be done in an adapter or a gateway. The architecture must support both synchronous and asynchronous messaging. Though the idea of Web services is

that client automatically adjusts to fit the service, this is unlikely to be the case all the times in the near future.

Routing Services

Routing services route the message to the right recipient. The messages are sent and delivered once and once only. In case the same service is provided by several systems, the routing service selects one of them in a way it has been programmed to function. To be able to select the targets, it maintains a searchable index (e.g. UDDI Registry) of all source and target applications and their locations. The registry also contains information on which protocols each service supports and how the services are initialised. The routing service contains information on Web service versions and their compatibilities, so that client using the older message format can be routed to the corresponding service.

Message Transformation Services

Message Transformation Services are used to make changes in the structure and content of the message. Different applications might have different views of the structure in which information is presented. Also, some systems use different types of product and customer numbers. The role of the Message Transformation Services is to enable communication of different systems, by converting the message to fit the recipients used data schema. The changes are mapped with metadata and processed by a transformation engine. Message transformation service must also recognise different message versions and transform the older request versions to fit the current service version.

Transaction Services

Transaction Services are used to support business processes and workflow. In case a transaction cannot be completed the transaction engine should notice this and (at least) assist the rollback. UPM-Kymmene's view was that the Transaction Services should just inform the requesting (client) application, which would then be responsible of rolling back the transaction. The Web services must be commit and rollback procedures to enable this functionality.

Security Services

Security Services was one of the most discussed issues during the interviews. The Security Services must enable encryption, authentication, authorisation, confidentiality, data integrity, and support digital certificates, and provide data privacy and non-repudiation of origin.

An important issue is that the client can rely that the service is provided by an trustworthy server and that the message content has not been changed. The broker should authenticate the human user, the computer and the software which does the request. Also the broker should check if the user has rights to request this particular service, including registry services.

Development Services

Development Services are used to maintain and develop transformation, routing and workflow rules. They should also help the documentation of these changes. Development services are also used to build new Web services and develop, publish and find service descriptions. The development tools must as well support the maintenance of different versions of service and messages, so that when developing the service the clients supporting the older service version can still use it.

The development environment must support simultaneous work of several people and have shared desktops and Concurrent Versions System (CVS) capabilities. The development tools should help the documentation of services and their interfaces and routing and workflow rules. Users should be able to search and publish services to third party service registries. Development tools should also enable developers at local sites to modify their service descriptions and allow partners to download published service descriptions to be used in their client development. It would be recommended that the development tools would use Eclipse platform for tool integration.

The integration architecture should have physically separated development, test and production environments. These separated components should ensure safe development and testing of transaction, transformation and routing rules as well as new Web services.

Runtime Services

Runtime Services provide the integration broker with load balancing, scalability and management capabilities. Though message loads can be estimated quite well the broker must survive of momentary peaks in traffic. The broker must be reliable, usable and transmit the messages quickly and correctly.

The runtime services provide support for load balancing, reliability, scalability, extensibility, and management services. Broker should monitor availability, usability, settings, messaging volumes, time, performance, targets and sources, service levels agreements. It should also audit the usage of Web services (see transactions) and alarm if something goes wrong. It should require minimum amount of human interaction when handling error situations, meaning the systems should be semiautomatic.

5.4.2 Adapter's Requirements

Adapters make it possible to fit systems in which do not support the broker's interfaces to the integration architecture. This section summarizes the requirements found from literature and interviews. As presented in sections 4.8 and 5.3, the adapter's requirements are divided into Adapter Services and Supporting Services. Adapter services perform the main functionality of an adapter. The adapter consists of an Application Programming Interface (API) for the integrated application and a common interface for the broker.

There should be two types of adapter, others for making Web service request and others for hosting Web services. The latter ones require HTTP listener capabilities, which can be provided by the WASP or the adapter itself. This requirement is kept in mind, but the requirements are presented just for one adapter, for simplicity reasons.

Connectivity Services

Connectivity Services take care of communication, addressing and delivery between the adapter and the broker. This is a Web service interface which uses a common transportation protocol, used both by the broker and the adapter. The protocol can be e.g. FTP, HTTP, HTTPR, HTTPS, POP, SMTP, WAP or MQ. In UPM-Kymmene's case it would probably be IBM MQ.

Interface Services

The Interface Services are used to integrate the adapter with the source or target application. Also the Interface Service can work as a gateway for server applications to connect the integration architecture through a certain interface. Adapters should be able to communicate with a wide range of application interfaces like C/C++, CORBA, Microsoft's COM+, Sun's J2EE, SAP's IDOC,

RPC, ABAP, Unix's RPC and IPC. Also, Interface Services should be able to communicate with other middleware and databases. Database connections should be done with native drivers or by using middleware, such as ODBC and JDBC. In addition, Interface Services should support Web service standards like UDDI, WSDL, XML and SOAP and the broker should be able to communicate with transaction engines like Tuxedo.

The Adapter should be able to present itself as a UDDI registry and a Web service, so that it can fool a proprietary system only supporting HTTP and SOAP by presenting itself as the service the application is looking for. This way the adapter hides the actual brokering infrastructure behind it and passes the requests in correct form to the broker for delivery.

Security Services

Adapter must support the security requirements set for the integration broker. Security Services enable encryption, authentication, authorisation, confidentiality, non-repudiation of origin, data privacy, data integrity, and support for digital certificates.

Adapter must support the authentication methods of the integration architecture and enable the systems behind them to authenticate the user. When the service cannot encrypt the message, the adapter should perform this.

Development Services

Development services are used to make changes to the adapters and their interfaces. Development Services should make reuse of adapters possible. Adapter's Development Services integrate with the broker's Development Services. Adapter must be able to be configured with parameters using the development tools and the updating should happen semi-automatically, so that

the same change can be done to several adapters at the same time. Adapter must retrieve the service descriptions from the centralized registry and configure itself to match this service and become a Web service client.

Runtime Services

Runtime Services are used to monitor the adapter and the service. They can follow the state of the service and how the security measures have been fulfilled. The adapter must be reliable, scalable and stable and work together with the broker and provide reliable messaging. Since adapters are not likely duplicated, they should be 100% reliable and if they are duplicated the architecture should be able to route requests to another adapter, if the primary one is unavailable.

5.5 Discussion on the Characteristics of Requirements

This chapter presents the requirements for a broker and an adapter. Probably these requirements do not contain all the necessary requirements, but they do give broad view of the environment and the issues needing to be covered. Also because of imbalanced answer rates this research area should be conducted in a larger scale.

UPM-Kymmene's business does not contain communication with consumers, if it did, the requirements might differ a bit. Also UPM-Kymmene does not have long transactions, which may change the characteristics of the integration architecture from a transaction engine to a more transaction supporting infrastructure, where transactions are rolled back by the clients.

The requirements found during the interviews emphasize the role of security, manageability, reliability and development. These were same issues, which

were not covered so well in the literature. The key requirements were that one must be sure who provides the service and that the messages are transmitted correctly. Use of existing MOM can provide a solution for the latter one. The security standards in Web services are still under development. The main reason for introducing broker architecture was the need to be able to maintain different versions of messages. Broker also makes it possible to communicate with partners who have a different message format in Web services or who use e.g. ebXML.

The following chapter describes a framework, which tries to address the requirements with the technologies both which were presented earlier in this master's thesis.

6 A FRAMEWORK FOR A BROKER BASED INTEGRATION ARCHITECTURE IN SOA

This chapter presents a framework for a broker based integration architecture in Service-Oriented Architecture. Service-Oriented Integration in a broker environment has not been covered in previous research. Though, it has been noted, that one can use message brokers to transport SOAP messages and that brokers will probably support SOI in the future. This framework presents in section 6.1 an overview of how an integration broker can be used in Service-Oriented Integration and what components the integration infrastructure should contain. The following sections drill into the components of the integration architecture and describe them in more detail and also present suitable standards to support functions of these components. Section 6.2 covers the components of the integration broker and the components of adapters are presented together with the WASPs in section 6.3.

This framework can be used to compare different integration brokers or it suggestion how SOI could be achieved. In this framework the broker is used e.g. to enable asynchronous communication and transactional integrity as well as to transform messages to fit the recipient. In addition, the broker enables the use of different communication protocols (e.g. have different protocols with partners than in intranet) and it is presumed that it would also ease the maintenance of SOI architecture.

6.1 Overview of the Integration Architecture

The integration architecture in this framework consists of a message broker and client and server type of adapters (see Figure 20). The message broker uses

primarily Web services for communication. It also supports several other protocols for communication. These protocols provide gateways for accessing the Service-Oriented Integration Architecture. Such gateways are used e.g. by partners communicating over leased lines or Internet, and users working with mobile terminals. As discussed earlier, two different types of adapters are required: ones for calling Web services and others for serving them. The server adapters require a HTTP-listener, which can either be in the adapter or provided by the platform the adapter runs on, which is a Web Application Service Platform. The client adapter can also run on the WASP platform, but it can also run directly together with the integrated application.

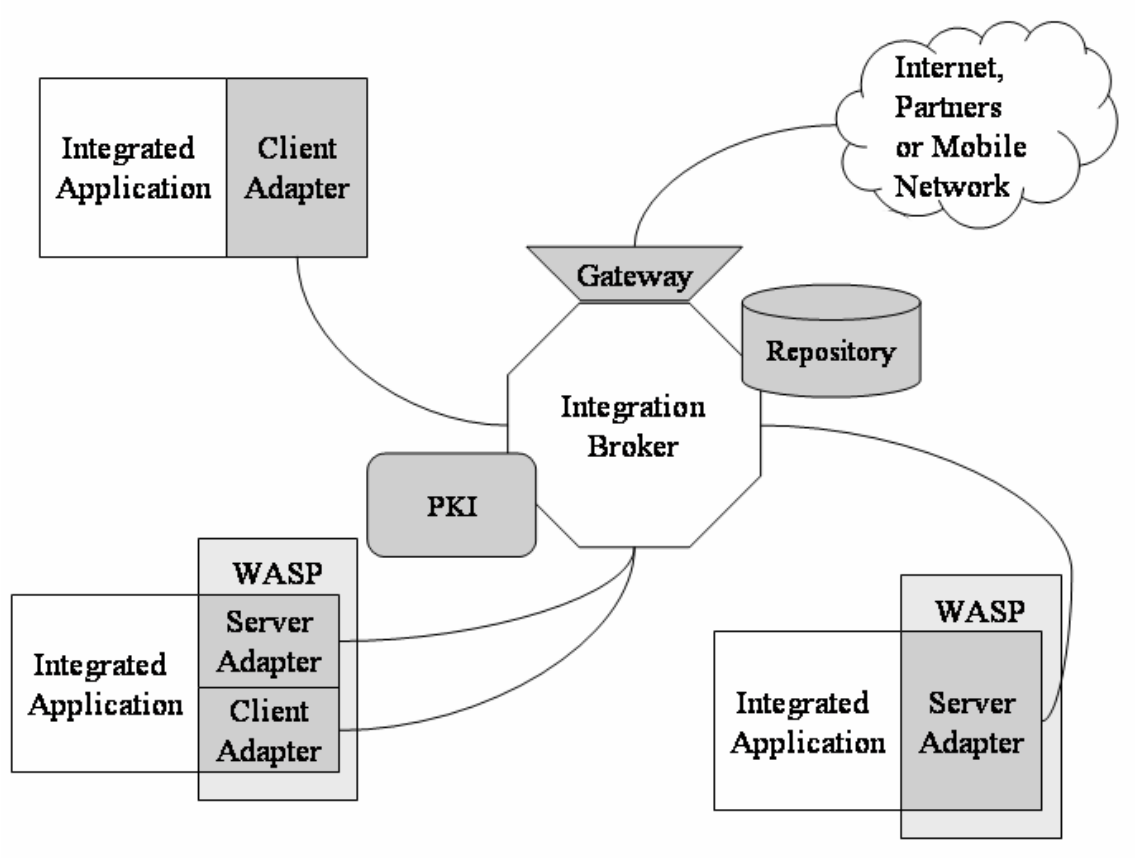


Figure 20. General overview of the integration architecture

The architecture also contains two other entities: connection to Public Key Infrastructure and a Repository. This research does not take an opinion on what

kind of a PKI system one should use, but a PKI is needed to be able to encrypt messages and authenticate clients and service providers. Both Web service clients, adapters and the broker use the same PKI. A repository is key element of the integration broker. A repository is a database holding information about e.g. message transformation, locations, security, design and architectural information, etc (Linthicum 2000a, 305-107).

The presented integration architecture uses Web services to provide the functionality of Service-Oriented Architecture. Though, Web services can be transmitted with also other protocols than HTTP, it will likely be de facto standard for proprietary applications and other Web service solutions. The use of a basic HTTP protocol contains a problem; it is a synchronous transport protocol. This problem can be tackled with the use of asynchronous transport mediums, such as HTTPR, JMS (Java Message Service), IBM MQSeries Messaging or MS Messaging. In addition to HTTP, HTTPS, RMI/IIOP and SMTP are synchronous transport protocols. (Adams 2000)

To provide asynchronous messaging the client has to be asynchronous as well. In a truly asynchronous messaging the requestor cannot stay waiting for minutes or days for the response. According to Adams (2000) there must be different threads for sending and receiving Web services and the Web service request itself must contain a reply-to address, so that the service can send the reply to the right address. There must also be a ***correlator***, which indicates which reply belongs to which request.

6.2 Integration Broker, Repository and Gateways

This section describes the parts of framework regarding the integration broker. The framework is based on the requirement division presented in section 4.8

and used in chapter 5. This division is based mainly on Kreger (2001) and Puschmann and Alt (2001). Now in this section this division is spiced with suitable standards relating to the different layers described in chapter 3.

The framework is illustrated in Figure 21. Each layer of the framework is described in its own section. The Connectivity Services in section 6.2.1, Routing Services in 6.2.2, and so on. Each section describes the primary functionality of that layer and also the standards used on that layer. At the moment there is a lack of industry wide standards for Web services. At the moment Web Services Interoperability organisation has only standardized the basic functionalities of Web services in Basic Profile 1.0. This specification covers SOAP 1.1, WSDL 1.1, UDDI 2.0, XML 1.0 and XML Schema, which are used in Connectivity Services (WS-I 2003). On the other layers therefore one has to presume that similar interoperability profiles will be published later. Since IBM and Microsoft together with their partners have been much more productive and devoted more for specifying Web service specifications, which they already are implementing, IBM's and Microsoft's specifications do have a strong role in this framework.

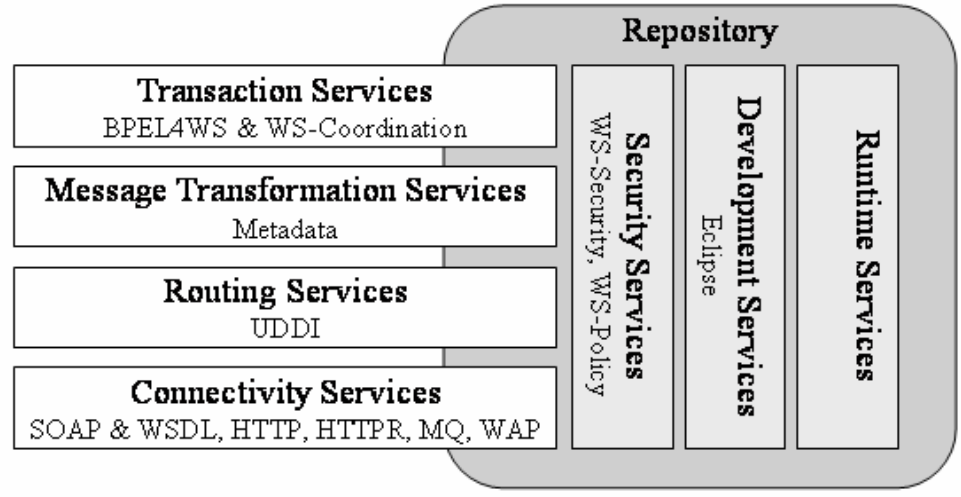


Figure 21. Architecture for an Integration Broker in Service-Oriented Architecture

The gateway's role in this section is very small. In this framework the broker provides gateways through its connectivity services by supporting several communication protocols. In the presentation of this framework it is presumed that the organisation will use existing MOM for internal message delivery and HTTPR for communication over the Internet. This selection is further discussed in section 6.2.2. The role of the repository is described in section 6.2.8.

6.2.1 Connectivity Services

The connectivity services take charge of communication, addressing and delivery. Connectivity services can be divided into transportation and XML based messaging layers. The transportation layer works as a gateway for other transportation protocols, such as WAP, HTTPR, FTP, POP and SMTP. It transforms communication basing on these protocols to the internal communication MOM protocol, in UPM-Kymmene's case IBM MQ.

The XML based messaging layer supports W3C's and WS-I's suggested standards for basic Web service communication. These standards were SOAP 1.1, WSDL 1.1, XML 1.0 and XML Schema (XSD). Note that DTD does not belong to this standard family.

6.2.2 Routing Services

The routing services are responsible of routing the message to the right recipient and they also host a service registry and control the policies, e.g. access policies. The architecture must support reliable messaging, both in synchronous and asynchronous forms. The communication network should also pass the ACID-test (Atomic, Consistent, Isolation and Durable).

Vivekanadan et al (2002) suggest the use of Message Oriented Middleware for providing asynchronous messaging capabilities. They also discuss that one

should establish a registry similar to UDDI, to describe which queue in MOM serves which service. (In MOM architecture messages are exchanged between the sender and recipient through queues.) Also during the interviews the willingness to use existing MOM was raised by several interviewees and several suppliers have indicated that MOM is an option for Web service transportation.

When communicating with partners, one must either use the existing middleware in collaboration with partners or use another reliable transport protocol. HTTPR suits these requirements well. It is layered protocol, which uses HTTP functionality (including e.g. SSL encryption), and extends them to enable reliable message transport. It also supports asynchronous messaging designed for Internet communication. (IBM 2002a. IBM 2002b)

Routing services also host a searchable UDDI registry, the UDDI 2.0 standard is the approved interoperable standard according to WS-I (2003). UDDI is used to publish and discover Web services and it contains the location of the technical definition of the Web service written in WSDL. WSIL (WS-Inspection) document can be used to help describing the same service in many different description languages; it contains references to all these descriptions and methods for finding the WSIL language.

The routing services use the policy standards (e.g. WS-Policy), which are hosted by security services. The routing services use security services to provide authentication schemes and other policies.

6.2.3 Message Transformation Services

Message transformation services transform the passed message payloads to fit the recipient's structural and data type requirements. The transformation services can be used e.g. to transform product IDs or transform the partner's

XML document structure to fit the organisation's own document structure. The transformation also helps version migrations, so that older clients can still use an updated Web service. The message transformation services use a dictionary (in this framework metadata) to describe the message formats that each application supports (Linthicum 2000a, 297). The metadata containing the transformation rules are defined in a graphical management environment and stored in a repository. Metadata, among other data related to the brokering activities, is stored in the repository.

6.2.4 Transaction Services

The transaction services provide methods for describing business processes and workflow. They also enable making an aggregation of Web services, so that a collection of Web service tasks is presented as a Web service. BPEL4WS language is used to define business processes and orchestrate Web services to perform the activities within a business process. WS-Coordination and WS-Transaction specifications are used together with BPEL4WS to define the exact methods and protocols how to e.g. several Web services can guide a process or how the transactional integrity is maintained.

In this framework transaction services are seen as a servant for managing message orders and correctness of processes. The client starting the process is responsible for acting correctly according to the information it receives from the transaction services. This means, that the client is responsible for committing the transactions and if an error is raised by the transaction services the client is responsible for doing a rollback. The transaction services assist the client, but the transaction service does not individually decide whether to commit or rollback.

6.2.5 Security services

Security services provide different security related services for the integration architecture. The security services can be divided into two different levels: Network security and XML Message Security (Kreger 2001, 24-25).

Network security

The network security can be provided with HTTPS, SSL and IPsec. These are methods for providing confidentiality and integrity on the message transport level. Network security can be used when there are no intermediaries, such situations can e.g. be when the broker is communicating with a service on the Internet. The problem with HTTPS, SSL and IPsec, which are the lower parts of the protocol stack, is that they can only be used when no intermediaries exist. (Selkirk 2001. Kreger 2001a, 24-25). Unfortunately, since the XML security standards have not been stabilised yet, network security can be the only usable method.

XML Message Security

XML security methods are methods for e.g. encrypting the payload of the SOAP message, not the whole SOAP message like in network security layer. Selkirk (2001) favours the use of XML standards for providing security, since XML makes supports portability, granularity and cross-utilisation of other XML standards.

The security standards of the Web services are still unaccomplished. Section 3.3.2 described possible future methods for providing Web service security. At the moment one has to rely on different methods, mainly HTTPS.

Web service standards aim at describing the use of encryption and authentication methods regardless of the used authentication technology or

product. This framework bases on the same ideology, the integration broker and the architecture must support the use of PKI for enabling XML message level security. However no requirement for the use of any specific PKI product or encryption method is presented. The PKI infrastructure must enable authentication, access control, confidentiality, integrity and non-repudiation as security services (Selkirk 2001). The architecture can use either internal PKI infrastructure or an external Web service based infrastructure for enabling encryption, authentication, digital certificates and other PKI based security measures.

Though there are no ready standards for the security of Web services, WS-Security is a promising specification. OASIS (2003) published a draft of its SOAP message security specification. Many different companies have contributed to the specification, therefore one could assume it has a good chance to become a widely implemented standard. Actually the SOAP message security specification is practically the same specification Microsoft and IBM published as WS-Security specification. WS-Security only addressed flexible support for multiple security token formats, multiple trust domains, multiple signature formats, and multiple encryption technologies. It does not address key derivation, advertisement and exchange of security policy or how trust is established or determined. These were addressed in other IBM's and Microsoft's specifications like WS-Trust. The fact that many of different organisations including Sun Microsystems is backing WS-Security, could be a good omen for other Microsoft and IBM based specifications such as WS-Policy, WS-Trust, WS-Privacy, WS-SecureConversation, WS-Federation and WS-Authorization. Therefore these specifications are seen as a base for the security services of this framework.

WS-Trust is a specification for describing Web service trust relationships between businesses. WS-Privacy specifies privacy policies and preferences in a Web service architecture. WS-SecureConversation describes methods for transmitting messages in a secure manner, when the messages are part of a larger business transaction. WS-Federation is a specification describing how incompatible security mechanisms are integrated. WS-Authorization describes methods for creating authorisation requests and decisions, for e.g. determining if a user has a right to access HR information. (Snell 2002)

Security services host also methods for managing Web service policies. These policies are described according to the WS-Policy model. WS-Policy together with its sub specifications (WS-PolicyAssertions and WS-PolicyAttachment) can be used to describe authentication schemes, transport protocol selection and other characteristics. Web services can retrieve policy and other metadata of a Web service through a Web service interface implemented according to the WS-MetadataExchange model.

Before WS-Security was published by OASIS, it was working on two specifications SAML (Security Assertion Markup Language) and XACML (Extensible Access Control Markup Language). SAML is for portable authentication and authorisation language and XACML is for standardising access control information. Verisign, Microsoft and Webmethods have their XML Key Management Specification (XKMS) for managing keys with Web services. (Selkirk 2001) The role of these specifications with WS-Security remains an open issue.

6.2.6 Development services

Development services are used to manage the brokering architecture and to develop Web services; they also help publishing and finding Web service

descriptions. Development services provide means to maintain and develop routing, transformation and business process rules. They also ease the documentation process of changes made to these rules.

The development environment should use Eclipse standard and support the work of several developers with CVS capabilities. The environment should also enable third party developers and partners to access Web service descriptions. The architecture must support physically separated development, test and production environments.

6.2.7 Runtime services

Runtime services make the integration broker scalable, reliable, and extensible and it is also capable of balancing message loads. They also provide means for monitoring availability, usability, settings, messaging volumes, time, performance, message targets and sources and service level agreements.

The system alerts the administration if an error occurs in any of the broker's components. For example if the transaction services note an error when performing a transaction it casts an alert.

6.2.8 Repository

Linthicum (2000a, 306) describes repository as a 'centre of the universe' in a message broker. The repository is a database in which security parameters, message schema information, metadata, transformation rules and a directory describing system locations are stored. Some of this information can be published or described with the standards presented in previous sections. For example, the specific location of WSDL document is described to Web services with UDDI, but UDDI is not a database containing this information. Similarly

as WS-Policy can describe authentication schemes, WS-Policy is not a method for storing the data. Repository would be the place to store this information.

Repository is in this sense the heart of the message broker, which enables CVS capabilities, makes it possible to run older clients with newer server applications by storing the transformation rules. It contains rules for message processing, stores metadata and other needed information. Still, repository is only a database storing the information, the broker itself uses the information stored in the repository to perform its tasks.

6.3 Adapters and WASPs

In this section the general architecture of the adapter and the functionality of a Web Application Service Platform (in section 6.3.3) are covered. As stated before there are two types of adapters: server and client side adapters. Server side adapters can use HTTP server from the WASP to enable Web service communication. The client side adapters are only required to be able to use HTTP POST and be able to receive SOAP over HTTP. Both process the SOAP request or replies in their own way. Otherwise the client and server adapters are similar and are presented here as one.

Figure 22 illustrates the architecture of an adapter. The adapter communicates with the Web Service Network through connectivity services. The adapter is connected to the integrated application through interface services. Similarly to the broker, adapter also hosts security services, development services and runtime services. For each service layer is described the standards that can be used to achieve its functionality. Following sections cover all the services and standards they should support.

Figure 22. Architecture of an Adapter in Service-Oriented Architecture

6.3.1 Connectivity Services and Interface Services

Connectivity services either host a Web service, described with WSDL or use one. The connectivity services use SOAP for receiving Web service requests and replying, or if adapter is a client it sends the SOAP requests as HTTP POST and receives them similarly as a Web browser. Connectivity services support HTTP, HTTPR and MQ as transport protocols. HTTP can be tunneled over MQ. The adapter also works as a buffer. If it is unable to send the message it will try to deliver it later. If service requests are sent asynchronously then there should be different threads for sending and receiving the messages and a coordinator for managing which service request belongs to which reply.

Interface services connect the adapter to the integrated application or database. It supports a wide range of application interfaces, such as CORBA, DCOM, EJB, IDOC, ABAP, RPC, Unix RPC and IPC, and native database drivers and database middleware such as ODBC and JDBC. The connectivity services should also support the communication between other middleware, such as transaction engines like Tuxedo.

The adapter provides means for integrating Web service capable proprietary software, which do not fit the integration architecture, by presenting itself as a UDDI registry and then as a Web service. This way the adapter can fool the software to use the adapter first as an UDDI registry then as the Web service. The adapter then reroutes the Web service and sends the request to the broker and to the correct recipient.

6.3.2 Runtime Services, Development Services and Security Services

Runtime services are used to monitor the adapter and the services they provide. Runtime services provide means for making the adapter scalable and extensible, also the adapter should have enough performance and maintain the integrity while making conversions and handling the possible exceptions. The adapter must also be 100% reliable. The management capabilities of the adapter should be integrated with the management capabilities of the actual broker.

Development services are used to change e.g. the Web service interfaces and the functionality of the adapter. The development tools of the adapter should integrate to the brokers' development tools. The adapters should also support batch processing so that the same configuration changes can be implemented at the same time to all the adapters.

Adapter's security services support the same security measures used by the broker. These rely mainly on WS-Security specification and other specifications closely related to it: WS-Policy, WS-Trust, WS-Privacy, WS-SecureConversation, WS-Federation and WS-Authorisation.

6.3.3 Web Application Service Platform

Adapters run on the Web Application Service Platform. They provide e.g. a HTTP server for hosting Web services. WASPS should be light and inexpensive

systems, because they are distributed in several locations. They should also support remote management.

The next chapter is the conclusions, in which, I will discuss the presented framework and the strengths of using a broker in Service-Oriented Integration. It also presents the future research issues under this topic.

7 CONCLUSIONS

The purpose of this master's thesis was to find requirements and present a framework for an integration broker in Service-Oriented Architecture. In this thesis is described different integration architectures and technologies that can be used to build a Service-Oriented Integration architecture. This master's thesis mainly concentrated on Web services and therefore the current and future Web service stacks were presented.

Requirements for the Service-Oriented Integration architecture were gathered both from literature and from a case study by interviewing UPM-Kymmene's employees. The interviews brought up some requirements unpublished in literature. Mainly the previously unpublished requirements concerned architecture's management tools, but also the smooth transition from Web service version to another. This requirement is premised on the scepticism that Web services could dynamically transform the SOAP request to fit the server's XML format. This master's thesis also covered adapters which have not been thoroughly researched. Also, based on the interviews, the term Web Application Service Platform was introduced to describe the platform needed for running server adapters (and client adapters).

The need of using Web services adapters is clear; one must be able to integrate non-Web service compliant systems to the integration architecture. The need of an integration broker can be discussed. However, there are advantages when using an integration broker in the SOI. First, the broker enables asynchronous communication and transactional integrity. The broker is a logical place for maintaining and following the order of messages. Organisations have existing experience of this with their Message Oriented Middleware. Also the broker is a good place for building workflow rules and creating aggregations of services.

The use of a broker makes it possible to build graphical user interface tools for creating these rules. Secondly, the broker is a tool for integrating different systems in the same architecture. The broker enables the transformation of message content and structure, which may vary because applications have e.g. different product numbers or partners have different views on the used structure of an invoice message. The broker also enables partners and organisation's mobile users to be connected to the integration infrastructure with gateways. In addition, broker provides an additional layer of defence, since it can block all the requests from a client, who is not authorized to call the target service. Thirdly, the maintenance advantage of broker architecture compared to point-to-point architecture is recognized. The use of broker eases the maintenance, since there are fewer connections between applications. It also provides a single place for maintaining communication rules between systems. Fourthly, the broker also provides a repository, which can store all the needed information in one place, which eases the management duties. Broker can host a UDDI registry, WSDL descriptions and other documents for describing policies and security settings.

One can see an analogue between the Service-Oriented Architecture and Message Oriented Architecture. In SOA the communication is bi-directional including request and response. In MOA the message is sent but any reply is rarely received. However, as presented in section 4.3, Web services should support request-response, publish-subscribe event and event-notification communication patterns. Web services are not restricted to request-response type of communication, SOAP can be used in one-way communication as in MOA and moreover Web services should support publish-subscribe communication, which is the traditional MOA communication method. The analogue between SOA and MOA means that organisations can provide MOA capabilities with SOA and vice versa. Of course this requires the SOA

components like UDDI and WSDL to exist, but organisations could use only one integration infrastructure to provide two types of integration. As it was stated in section 4.2, Web services will just be another service in an integration broker. It is highly possible that SOAP will be the next lingua franca in EAI. SOAP can transport any kind of message payload and the broker can transform the payload to fit the recipient. Probably organisations will start transforming from proprietary MOA message formats to SOAP and gradually, as EAI products progress, move to Service-Oriented Integration.

There should no reason why the presented framework would not be universally applicable. Though, UPM-Kymmene is not an organisation that is so dependent on transactions, this does not show in the framework. The basis of the presented framework is formed by other frameworks, standards and specifications not designed for UPM-Kymmene. Also, there was no conflict between the requirements from the case study and literature, moreover I would state they complemented and verified each other. Therefore I claim that this framework would be applicable also in other organisations. The testing of this framework will be an issue one has to study further.

Some problems points can be seen in this study. Firstly, the described standards and specifications are not analyzed in depth, nor compared. Also Microsoft and IBM are heavily involved in all of the standards used. Some counter arguments are, that standards like DAML-S have not received much interest in any vendor, so if one should build a SOI architecture using such standards it would not be interoperable. Secondly, Sun Microsystems and its partners have published very few Web service specifications and those have been deemed incomplete. At the moment IBM and Microsoft are producing specifications in much more comprehensive way. The fact that IBM and Microsoft submitted their WS-Security standard for OASIS and it was accepted, tells that neither IBM or Microsoft want to create their own proprietary standards, but do want open

standards. Therefore it is highly possible that the other IBM's and Microsoft's specifications will be submitted to OASIS and accepted. The backing of major IT vendors would seem to be a good basis for industry wide Web service standards.

One weakness with this framework is that it is heavily based on non-scientific models. Though the parts covering integration brokers and EAI reference architecture are scientifically accepted, the resulting framework bases heavily on corporate references regarding Web services stacks and standards. Still, IBM's conceptual Web service architecture (Kreger 2001) is one of most referenced of Web services related research. At the moment there is very little research on standards and – excluding Ankolekar et al (2001) suggestion on DAML-S – no scientific proposals on Web service architectures. Therefore, since all the development work with Web services is emphasized in companies there was no other choice than use material from corporate research.

This framework can be used to compare different integration brokers supporting SOA. It also presents a view how the specification process might go further and describes the current state of Web services standardisation process. In the future it could be worthwhile to test the framework with brokers supporting SOA and test the framework in different types of organizations if it suites their requirements.

REFERENCES

Adams H. 2002. Asynchronous operations and Web services, Part 1: A primer on asynchronous transactions. BM developerWorks [online]. [referred 31.10.2003] Available at <http://www-106.ibm.com/developerworks/webservices/library/ws-asynch1.html>.

Ankolekar A., Burstein M., Hobbs J. R., Lassila O., Martin D. L., McIlraith S. A., Narayanan S., Paolucci M., Payne T., Sycara K. & Zeng H. 2001. DAML-S: Semantic markup for Web services. Proceedings of the International Semantic Web Working Symposium, Stanford, California [online]. [referred 22.5.2003] Available at < http://www.daml.org/services/SWWS.pdf>.

Arsanjani A., Hailpern B, Martin J., Tarr P. 2003. Web Services – Promises and Compromises. ACM Queue, March 2003, 1(1), 48-58.

BEA, IBM & Microsoft. 2003a. Web Services Addressing (WS-Addressing). MSDN Library [online]. [referred 26.10.2003] Available at <http://msdn.microsoft.com/webservices/default.aspx?pull=/library/en-us/dnglobspec/html/ws-addressing.asp>.

BEA, IBM & Microsoft. 2003b. Web Services Coordination (WS-Coordination). MSDN Library [online]. [referred 26.10.2003] Available at <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/wscoor.asp>.

BEA, IBM & Microsoft. 2003c. Web Services Atomic Transaction (WS-AtomicTransaction). MSDN Library [online]. [referred 26.10.2003] Available at

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/wsat.asp>.

BEA, IBM, Microsoft & SAP. 2003a. Web Services Policy Framework (WS-Policy). MSDN Library [online]. [referred 26.10.2003] Available at <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-policy.asp>.

BEA, IBM, Microsoft, SAP. 2003b. Web Services Policy Assertions Language (WS-PolicyAssertions). MSDN Library [online]. [referred 26.10.2003] Available at <http://msdn.microsoft.com/msdn-online/shared/components/ratings/ratings.aspx?ContentID=_921181&HideDiscuss=1>.

BEA, IBM, Microsoft & SAP. 2003c. Web Web Services Policy Attachment (WS-PolicyAttachment). MSDN Library [online]. [referred 26.10.2003] Available at <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-policyattachment.asp>.

BEA, IBM, Microsoft, SAP & Siebel Systems. 2003. Specification: Business Process Execution Language for Web Services Version 1.1. IBM developer Works [online]. [referred 27.10.2003] Available at <http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/>.

BEA, IBM, Microsoft & TIBCO. 2003. Web Services Reliable Messaging Protocol (WS-ReliableMessaging). MSDN Library [online]. [referred 26.10.2003] Available at <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-reliablemessaging.asp>.

Beveridge T. 2000. Building EAI Adapters for Legacy Systems. The Journal of Object-Oriented Programming 13(4), 2-5.

Bechini A., Foglia P. & Prete C.A. 2002. Use of a CORBA/RMI gateway: characterization of communication overhead. In Proceedings of the third international workshop on Software and performance. Rome, July 24-26 2002, ACM Press, New York 2002, 150-157.

Birrell A. & Nelson B.J. 1984. Implementing Remote Procedure Calls. ACM Transactions on Computer Systems, February 1984, 2 (1), 39-59.

Brittenham P. 2002. An overview of the Web Services Inspection Language. IBM developerWorks [online]. [referenced 23.10.2003] Available at <http://www-106.ibm.com/developerworks/webservices/library/ws-wsilover/>

Brown A.W., Feiler P.H. & Wallnau K.C. 1992. Past and Future Models of CASE Integration. In Proceedings of the 5th International Workshop on CASE (CASE'92), Montreal, Canada, July 6-10. Los Alamitos: IEEE, 36-45.

Delphi Group. 2002. Web Services 2002 - Market Milestone Report. [online] [referenced 25.6.2003] Available at <http://www.delphigroup.com/research/whitepapers/2002-ws-market-milestone.pdf>

Eclipse. 2003. Consortium [online]. [referred 16.9.2003] Available at <http://www.eclipse.org/org/index.html>.

Editors of The American Heritage (Eds). 2000. The American Heritage Dictionary of the English Language, Fourth Edition. Boston: Houghton Mifflin Company.

Elfatatry A. Layzell P.J. 2003. Negotiating In Service Oriented Environments. Accepted for publication by Communications of the ACM [online]. [referred 27.6.2003] Available at <http://service-oriented.com/publications/Negotiation.pdf>.

Ferguson D.F., Storey T., Lovering B., Shewchuk J. 2003. Secure, Reliable, Transacted Web Services: Architecture and Composition. MSDN Library [online]. [referred 25.10.2003] Available at <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwebsrv/html/wsoverview.asp>.

Fujitsu, Hitachi, NEC, Oracle, Sonic & Sun. 2003. Oracle Technology Network [online]. [referred 27.10.2003] Available at <http://otn.oracle.com/tech/webservices/htdocs/spec/WS-ReliabilityV1.0.pdf>.

Gibb B. & Damodaran S. 2003. ebXML: Concepts and Application. Indiananapolis: Wiley Publishing, Inc.

Gisolfi D. 2001. Web service architect, Part 3: Is Web services the reincarnation of CORBA? IBM developerWorks [Online]. [referred 13.10.2003] Available at <http://www-106.ibm.com/developerworks/webservices/library/ws-arc3/>.

Gold-Bernstein B. 1999. EAI Market Segmentation, EAI Journal, July/August.

Gold-Bernstein B. 2003. Technology Vs. Architecture, ebizQ [online]. [referred 10.6.2003] Available at <http://eai.ebizq.net/str/goldbernstein_7.html>.

Gokhale A., Kumar B. & Sahuguet A. 2002. Reinventing the Wheel? CORBA vs Web services. Practice and Experience Track, Eleventh International Conference on World Wide Web (WWW2002), Honolulu, Hawaii, May 7-

11 [online]. [referred 22.5.2003] Available at <http://www2002.org/CDROM/alternate/395/>.

Gottschalk K., Graham S., Kreger H., Snell J. 2002.Introduction to Web service architecture. IBM Systems Journal, 41 (2), 170-177.

Hildreth S. 2000. Smart Adapters: How New Adapter Technologies Can Provide a Scalable, More Distributed Integration Architecture. ebizQ [online]. [referred 28.10.2003] Available at <http://www.ebizq.net/topics/adapters/features/1604.html>.

Hirsijärvi S. & Hurme H. 2001. Tutkimushaastattelu – Teemahaastattelun teoria ja käytäntö. Helsinki: Helsinki University Press.

IBM. 2002a. HTTPR Specification. IBM developerWorks [online]. [referred 31.10.2003] Available at <http://www-106.ibm.com/developerworks/webservices/library/ws-httprspec/>

IBM. 2002b. A Primer for HTTPR - An overview of the reliable HTTP protocol. IBM developerWorks [online]. [referred 31.10.2003] Available at <http://www-106.ibm.com/developerworks/webservices/library/ws-phtt/>.

IBM & Microsoft. 2002. Security in a Web Services World: A Proposed Architecture and Roadmap. MSDN Library [online]. [referred 26.10.2003] Available at <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwssecur/html/securitywhitepaper.asp>

IBM, Microsoft, RSA Security & VeriSign. 2002a. Web Services Trust Language (WS-Trust). MSDN Library [online]. [referred 26.10.2003] Available at <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-trust.asp>.

IBM, Microsoft, RSA Security & VeriSign. 2002b. Web Services Secure Conversation Language (WS-SecureConversation). MSDN Library [online]. [referred 26.10.2003] Available at <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-secureconversation.asp>.

IBM, Microsoft, RSA Security & VeriSign. 2003. Web Services Federation Language (WS-Federation). MSDN Library [online]. [referred 26.10.2003] Available at <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-federation.asp>.

Johannesson P., Jayaweera P. & Wangler B. 2000. Application and Process Integration – Concepts, Issues and Research Directions. In S. Brinkkemper, E. Lindencrona and A. Solvberg (eds.), Information Systems Engineering: State of the Art and Research Themes, London: Springer, 159-169.

Johannesson P & Perjons E. 2001. Design Principles for Process Modelling in Enterprise Application Integration. Information Systems 26(3), 165-184.

Jennings R. 2003. SOAP Extension Soup. XML & Web Services Magazine [online]. [referred 27.10.2003] Available at <http://www.fawcette.com/xmlmag/2003_01/online/webservices_rjennings_01_31_03/>.

Khalaf R., Mukhi N. & Weerawarana S. 2003. Service-Oriented Composistion in BPEL4WS. In Proceedings of The Twelfth International World Wide Web Conference, Budapest, Hungary, May 20-24 [online]. [referred 27.10.2003] Available at <http://www2003.org/cdrom/papers/alternate/P768/choreo_html/p768-khalaf.htm>.

Kreger H. 2001. Web Services Conceptual Architecture. International Business Machines Corporation, IBM Software Group, Report WSCA 1.0.

Kuebler D. & Eibach W. 2002. Adapting legacy applications as Web services. IBM developerWorks [online]. [referred 28.10.2003] Available at <http://www-106.ibm.com/developerworks/webservices/library/ws-legacy/>.

Lang U. 2003. Access policies for middleware. University of Cambridge, Computer Laboratory, Technical Report UCAM-CL-TR-564.

Leymann F. 2001. Web Services Flow Language (WSFL 1.0) [online]. IBM Software Group, IBM Academy of Technology. [Referred 15.10.2003] Available at <http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf>.

Leymann F. Roller D. & Schmidt M.-T. 2002. Web services and business process management. IBM Systems Journal, 41 (2), 198-211.

Linthicum D. 2000a. Enterprise Application Integration. Boston: Addison-Wesley

Linthicum D. 2000b. EAI Application Integration Exposed. Software Magazine, February/March 2000, 48-52.

Microsoft. 1996. DCOM Technical Overview [online]. [referred 30.9.2003] Available at <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndcom/html/msdn_dcomtec.asp>.

Microsoft. 2003. Web Services Specifications. MSDN Library [online]. [referred 30.9.2003] Available at

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/wsspecsover.asp>.

Myerson J.M. 2002. Web Services Architectures. In P. Fletcher & M. Waterhouse (eds.) Web services Business Strategies and Architectures. Birmingham: Expert Press Ltd, 38-54.

OASIS. 2003- Web Services Security: SOAP Message Security - Working Draft 17, Wednesday, 27 August 2003 [online]. [referred 1.11.2003] Available at <http://www.oasis-open.org/committees/documents.php>.

Object Management Group. 2002. CORBA Basics. [online]. [referred 30.9.2003] Available at <http://www.omg.org/gettingstarted/corbafaq.htm>.

Object Management Group. 2001. ORB Basics. [online]. [referred 30.9.2003] Available at <http://www.omg.org/gettingstarted/orb_basics.htm>.

Puschmann T. & Alt R. 2001. Enterprise Application Integration - The Case of the Robert Bosch Group. In Proceedings of the 34th Hawaii International Conference on System Sciences, Maui, January 3-6. Los Alamitos: IEEE Computer Society.

Papazoglou M.P. 2003. Web Services and Business Transactions. World Wide Web: Internet and Web Information Systems 6 (1), 49–91.

Samtani G. & Sadhwani D. 2002a. Enterprise Application Integration (EAI) and Web Services. In P. Fletcher & M. Waterhouse (eds.) Web services Business Strategies and Architectures. Birmingham: Expert Press Ltd, 38-54.

Samtani G. & Sadhwani D. 2002b. Integration Brokers and Web Services. In P. Fletcher & M. Waterhouse (eds.) Web services Business Strategies and Architectures. Birmingham: Expert Press Ltd, 70-82.

Selkirk A. 2001. Using XML security mechanisms. BT Technol Journal 13(3), 35-43.

Snell J. 2002. Securing Web services. IBM [online]. [referred 1.10.2003] Available at <http://www-3.ibm.com/software/solutions/webservices/pdf/wp_securingws.pdf>.

Sollazzo T., Handschuh S., Staab S., Frank M. & Stojanovic N. 2002 Semantic Web Service Architecture — Evolving Web Service Standards toward the Semantic Web. In S. M. Haller, G. Simmons (eds.) Proceedings of the 15th International FLAIRS Conference, Pensacola, Florida, May 16-18. Menlo Park: AAAI Press.

Sun Microsystems. 2003. Enterprise JavaBeans Specification,Version 2.1. [online] [referred 1.10.2003] Available at <http://java.sun.com/products/ejb/docs.html>.

Tombros D., Geppert A. & Dittrich K.R. 1995. SEAMAN: Implementing Process-Centered Software Development Environments on Top of an Active Database Management System. Universität Zürich, Institut für Informatik, Technical Report 95.03.

Truelove K. 2001. Web services networks - Intermediaries that simplify inter-enterprise projects [online]. [referred 25.6.2003] Available at <http://www-106.ibm.com/developerworks/library/ws-netwrk.html>.

Tsalgatidou A. & Pilioura T. 2002. An Overview of Standards and Related Technology in Web Services. International Journal of Distributed and Parallel Databases, Special Issue on E-Services, 12(2), 135-162.

uddi.org. 2000. UDDI Technical White Paper [online]. [referred 12.10.2003] Available at <http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf>.

UPM-Kymmene. 2003. Annual Report 2002 [online]. [referred 10.11.2003] Available at <http://w3.upm-kymmene.com/gho/internet/ghofinre.nsf/c310deb8b0d11b51c2256949001fd4bc/42e4301ee2658271c2256ce10052e5e1/$FILE/AR_2002_en.pdf>.

Vivekanandan S.M., Tso K.K.K., Thompson M.K. & De Roure D.C. 2002. Asynchronous Linking in a Service Oriented Architecture. In D. Millard, J.M. Haake, S. Reich (eds.) Proceedings of the International Workshop on Open Hypermedia Systems Core Concepts & Research Directions - Pre-Conference Workshop at the ACM 13th International Conference on Hypertext and Hypermedia (HT'02), College Park, Maryland, June 12. FernUniversität, Fachbereich Informatik, 295 – 8/2002 [online]. [referred 31.10.2003] Available at <http://www.ecs.soton.ac.uk/~dem/workshops/ohs2002/ohs2002-proceedings.pdf>

W3C. 2000. Simple Object Access Protocol (SOAP) 1.1. Box D., Ehnebuske D., Kakivaya G., Layman A., Mendelsohn N., Nielsen H. F., Thatte S. & Winer D. [online]. [referred 12.10.2003] Available at <http://www.w3.org/TR/SOAP/>.

W3C. 2001. Web Services Description Language (WSDL) 1.1. E. Christensen, Curbera F., Meredith G., Weerawarana S. [online]. [referenced 12.10.2003] Available at <http://www.w3.org/TR/wsdl>.

W3C. 2002a. Web Services Architecture Requirements. W3C Working Draft 14 November 2002. D. Austin, A. Barbir, C. Ferris & S. Garg (eds.) [online].

[referenced 12.5.2003] Available at <www-format http://www.w3.org/TR/2002/WD-wsa-reqs-20021114/>.

W3C. 2002b. Web Services Glossary. W3C Working Draft 14 November 2002. A. Brown, H. Haas (eds.). [online] [referenced 12.5.2003] Available at <www-format <http://www.w3.org/TR/2002/WD-ws-gloss-20021114/>.

W3C. 2002c. Web Services Architecture. W3C Working Draft 14 November 2002. M. Champion, C. Ferris, E. Newcomer and D. Orchard (eds) [online]. [referred 23.10.2003] Available at <http://www.w3.org/TR/2002/WD-ws-arch-20021114/>

W3C. 2003a. SOAP Version 1.2. Part 0: Primer. N. Mitra (ed.). [online] [referred 12.10.2003] Available at < http://www.w3.org/TR/soap12-part0/>.

W3C. 2003b. SOAP Version 1.2 Part 1: Messaging Framework. M. Gudgin, M. Hadley, N. Mendelsohn, J-J. Moreau, H. F. Nielsen (eds) [online] [referred 12.10.2003] Available at < http://www.w3.org/TR/soap12-part1/>.

W3C. 2003c. Web Services Architecture. W3C Working Draft 8 August 2003. D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris and D. Orchard (eds) [online]. [referred 23.10.2003] Available at <http://www.w3.org/TR/ws-arch/>.

Wangler B. & Paheerathan S.J. 2000. Horizontal and Vertical Integration of Organizational IT Systems. In S. Brinkkemper, E. Lindencrona, and A. Sölvberg (eds) Information Systems Engineering: State of the Art and Research Themes, June 5-6 2000, Stockholm. Heidelberg: Springer-Verlag.

Wap Forum. 2002. Wireless Application Protocol WAP 2.0 Technical White Paper. [online] [referenced 25.9.2003] Available at <http://www.wapforum.org/what/WAPWhite_Paper1.pdf>.

WS-I. 2003. WS-I Announces General Availability of the Basic Profile 1.0. Press Release [online]. [referenced 27.10.2003] Available at <http://www.ws-i.org/docs/20030812wsipr.htm>.

Wohed P., van der Aalst W.M.P., Dumas M. & ter Hofstede A.H.M.. 2002. Pattern-Based Analysis of BPEL4WS. Queensland University of Technology, Faculty of Information Technology, Technical Report FIT-TR-2002-04.

Recorded Interviews

Interviewee 1. 8.8.2003. Kuusankoski, Finland.

Interviewee 2. 14.8.2003. Helsinki, Finland.

Interviewee 3. 15.8.2003. Helsinki, Finland.

Interviewee 4. 15.8.2003. Helsinki, Finland.

Interviewee 5. 15.8.2003. Helsinki, Finland.

Interviewee 6. 20.8.2003. Kuusankoski, Finland.

Interviewee 7. 21.8.2003. Lappeenranta, Finland.

Interviewee 8. 21.8.2003. Lappeenranta, Finland.

APPENDIX 1. INTERVIEW MATERIAL (IN FINNISH)

HAASTATTELUMATERIAALI

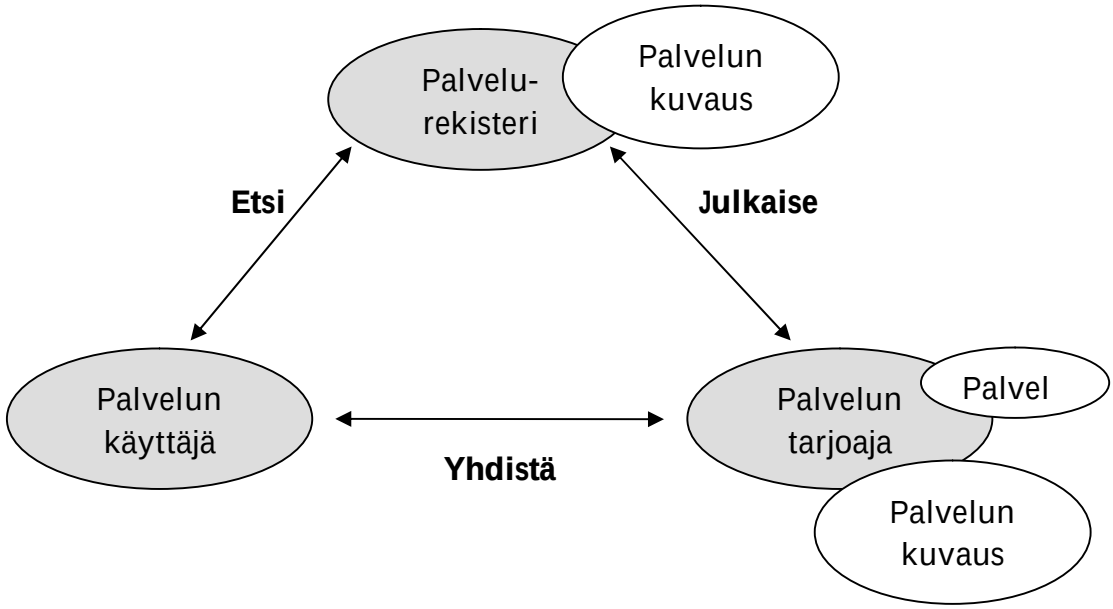
Tämän haastattelun tarkoituksena on selvittää vaatimuksia joita palvelusuuntautunut arkkitehtuuri asettaa järjestelmäintegraatiolle. ***Palvelusuuntautunut arkkitehtuuri*** (Service-Oriented Architecture, SOA) on tapa toteuttaa järjestelmäintegraatio siten, että järjestelmät tarjoavat palveluita toisilleen, esim. valuuttakurssimuunnospalvelu. Palveluun liittyy palvelun pyyntö ja vastaus, jotka tapahtuvat useimmiten lyhyen ajan sisällä.

Tällä hetkellä järjestelmäintegraatio toteutetaan viestinvälitystä hyödyntäen. Se ei tarjoa samanlaisia mahdollisuuksia tarjota palveluita muille järjestelmille ja partnereille.

Palvelusuuntautunut arkkitehtuuri (SOA) ei ole mikään uusi ajatus. Käsitteenä se on ollut olemassa jo komponenttitekniikoiden yhteydessä. Se on tietyssä mielessä vain jatkoa komponenttitekniikoille. SOA-arkkitehtuuri on noussut viime vuosien aikana keskusteluihin web-palveluiden (Web services) vuoksi, joiden avulla voidaan toteuttaa SOA-arkkitehtuuri alustariippumattomasti.

SOA-arkkitehtuurin ajatuksena on se, että järjestelmät tarjoavat toisilleen hyvin määritellyn rajapinnan kautta toisilleen palveluita. Näiden palveluiden kautta järjestelmiä voidaan integroida toisiinsa. SOA tarjoaa mahdollisuuden dynaamiseen järjestelmäintegraatioon.

SOA-arkkitehtuuri koostuu kolmesta komponentista, palvelun tarjoajasta, palvelun pyytäjästä ja palvelurekisteristä (ks. Kuva 23).



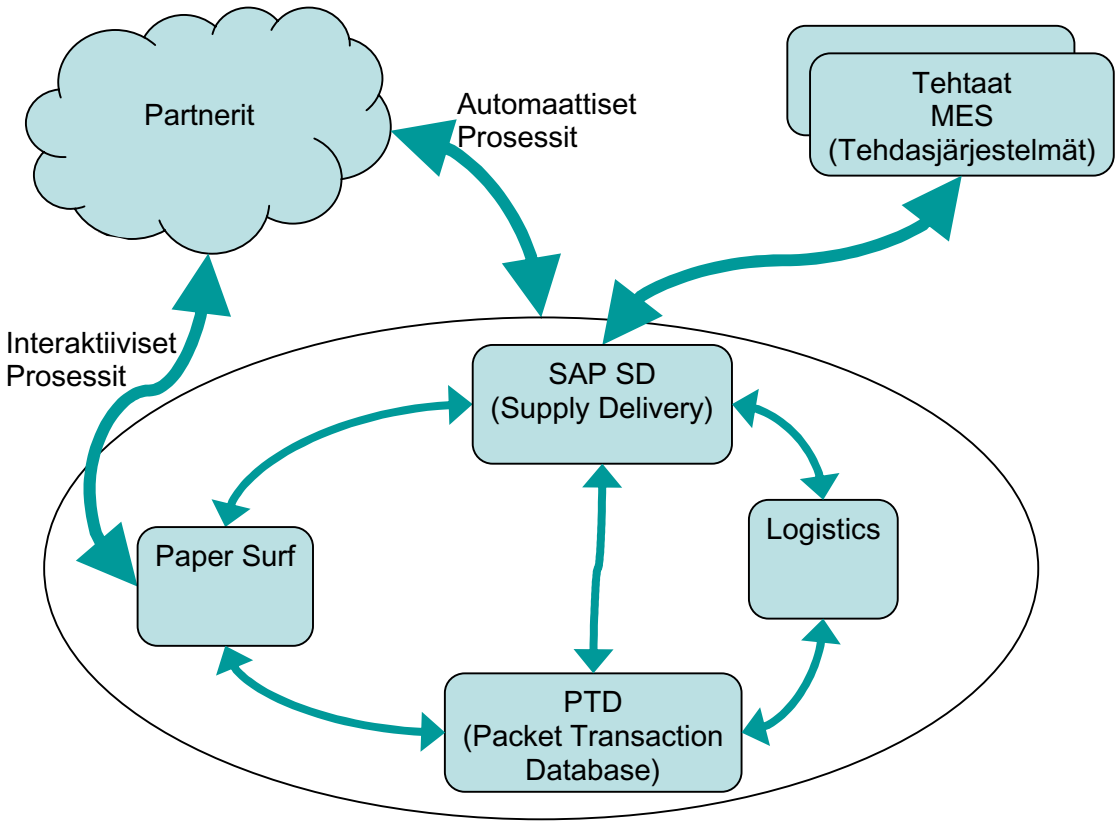
Kuva 23. Palvelusuuntautunut arkkitehtuuri ja sen komponentit

Palvelun tarjoaja on alusta, joka ylläpitää palvelua muiden käyttöön. Palvelun olemassa olosta kerrotaan julkaisemalla se palvelurekisteriin. Palvelurekisteri sisältää kuvaukset olemassa olevista palveluista ja miten niihin saa yhteyden. Palvelun käyttäjä etsii sopivaa palvelun palvelurekisteristä. Löydettyään sopivan palvelun käyttäjä kutsuu palvelua tavalla, joka on määritetty palvelun kuvauksessa. Palveluntarjoaja vastaa palvelun pyyntöön. Vastauksen muoto on myös määritetty palvelun kuvauksessa.

Palvelusuuntautunut arkkitehtuuri on usein yhdistetty web-palvelutekniikkaan, joskin se voidaan toteuttaa myös muilla ’ohjelmointikielillä’. Palvelunkutsut ja vastaukset voidaan välittää millä tahansa protokollalla ja myös väliohjelmistoja (middleware) ja sanomanvälittäjiä (message broker) voidaan käyttää.

Alla on kuvattu UPM:n toimitusketjunhallinnan järjestelmiä, jotka eivät ehkä tue sellaisenaan palvelusuuntautunutta arkkitehtuuria. Liittyen ko. ympäristöön ja järjestelmiin pohdi tilannetta, joissa järjestelmät voisivat tarjota

toisilleen palveluita. Valitse neljä eri tilannetta, joissa järjestelmät voisivat tarjota toisilleen palveluita. Mikäli mahdollista valitse sellaisia tapauksia, että tapaukset olisivat laajempia kuin pelkkä kysely ja sen vastaus. Miten palvelusuuntautuneen arkkitehtuurin käyttö näissä tilanteissa vaikuttaisi järjestelmäintegraatioon ja minkälaisia vaatimuksia näistä tilanteista nousisi?



Kuva 24. UPM:n toimitusketjunhallinnan järjestelmiä

Liittyen kuhunkin näihin neljään tapaukseen liittyen voit pohtia seuraavia seikkoja ja mitä erityistä olisi huomioitava näissä tapauksissa: (1) yhteenliitettävyys, (2) luotettavuus, (3) tietoturva, skaalautuvuus ja laajennettavuus, (4) integraatio WWW:n kanssa, (5) vaatimukset käyttäjiltä (6) hallinnointi (7) transaktionhallinta ja asiankäsittely (workflow) sekä (8) järjestelmäkehitys.

Haastattelutehtävät

Liittyen UPM:n toimitusketjunhallinnan järjestelmiin pohdi tilannetta, joissa järjestelmät voisivat tarjota toisilleen palveluita. Valitse neljä eri tilannetta, joissa tätä toiminnallisuutta voitaisiin käyttää järjestelmien integrointiin. Pyri löytämään tapauksia, jotka ovat laajempia kuin pelkkä kysely ja vastaus.

Lisäksi voit pohtia kuhunkin tilanteeseen liittyen yhteenliitettävyyttä, luotettavuutta, tietoturvaa, skaalautuvuutta ja laajennettavuutta, integraatiota WWW:n kanssa, käyttäjien asettamia vaatimuksia, hallinnointia, transaktionhallintaa ja asiankäsittelyä (workflow), sekä järjestelmäkehityksen asettamia vaatimuksia.

Minuun voi ottaa yhteyttä, mikäli materiaali tuntuu epäselvältä tai muuten vain haluatte tarkennuksia. Minuun saa yhteyden sekä sähköpostitse että puhelimitse.

Kiitoksia!

-Tuomas Vanhanen