WEB SERVICE ORCHESTRATION STANDARDS

by

Çiğdem Patlak

B.S. in C.E., Yıldız Technical University, 1999

Submitted to the Institute of Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science
In
Computer Engineering

Boğaziçi University

2003

ACKNOWLEDGEMENTS

Many people have supported me throughout this thesis study and I would like to express my sincere gratitude. First of all I want to thank my thesis advisor Dr. Haluk Bingöl for his good advice and useful hints, and for keeping my attention focused. I am also thankful to Dr. Ayşe Başar Bener and Dr. Selim Akyokuş for their continous support. Special thanks go to Vladimir Tosic (a Ph.D. candidate in Carleton University), who first came up with this thesis topic and provided help plus motivation for my research.

I also have to mention my family, for all those small but valuable things that made this study possible. A big-thank-you goes to Volkan for his encouragement throughout the research work and the preparation of this thesis.

My friends and colleagues in Dışbank should also be thanked, as they showed patience and understanding during the writing of this thesis. Some of them provided technical guidance and advise, whereas some gave me courage to complete this work. I also want to mention the help of Attila Paternoster, Application Software Manager in Wincor-Nixdorf Turkey, for providing numerous technical sources and literature.

Last but not least, a word of thanks goes to Fazıl Say for his inspiring piano-playing.

ABSTRACT

WEB SERVICE ORCHESTRATION STANDARDS

Web services are a standards-based, loosely coupled new distributed computing architecture designed and specified to foster cross-platform program-to-program communications. Currently the Web services landscape is in an evolving state with core specifications almost mature, whereas more specific standards for complex business requirements have not been finalized. The principal goal underlying this thesis is to highlight the potential of Web service solutions as an enabling distributed computing technology by studying the role, standards and deficiencies of Web service technologies. The thesis study includes a state-of-the-art study on Web services technology and its radical changes on application innovation and development. Due to competing vendor specifications in the field of "Web Service Orchestration", a major part of the study is devoted to taking different viewpoints on existing business process languages such as BPEL4WS, WSCI and BPML so as to produce an extensive comparative study. As a general scheme for Web service adoption does not exist, the work is concluded with a financial Web service use case to investigate the feasibility of the Web services paradigm for application development in the financial services industry.

ÖZET

WEB SERVİS ORKESTRASYON STANDARTLARI

Web servisleri internet standartlarını temel alan, farklı ortamlardaki uygulamaların ortak calışmasını sağlamak üzere tasarlanmış olan yeni bir dağıtık yazılım mimarisidir. Günümüzde web servis teknolojilerinin gelişim süreci devam etmektedir. Temel standartlar ile ilgili calışmalar tamamlanmış olmakla birlikte, karmaşık iş gereksinimlerini karşılayacak spesifik standartların hazırlığı henüz sonlanmamıştır. Bu tezin hazırlanmasındaki başlıca hedef, web servis teknolojilerinin rolünü, standartlarını ve eksikliklerini inceleyerek, web servislerin dağıtık yazılım çözümü olma potansiyelini ortaya koymaktır. Tez çalışması web servis teknolojileri ile ilgili kapsamlı inceleme sonuçları içermektedir. Özellikle "Web Servis Orkestrasyon" alanında sayısız standart yayınlandığından, çalışmanın büyük bir bölümü bu konuyla ilgili olarak hazırlanan BPEL4WS, WSCI ve BPML gibi dilleri araştırmak ve karşılaştırmalı sonuçlar üretmek için ayrılmıştır. Web servis teknolojilerine geçiş için genel bir plan mevcut olmadığından, örnek bir finans web servisi hazırlanarak finans servisleri açısından Web servis uygulama geliştirme olanakları araştırılmıştır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ADS	Advertisement and Discovery of Services
API	Application Programming Interface
ASP	Application Services Provider
B2B	Business-to-Business
B2C	Business-to-Consumer
BPEL4WS	Business Process Execution Language for Web Services
BPM	Business Process Management
BPML	Business Process Modeling Language
BPMN	Business Process Modeling Notation
BPQL	Business Process Query Language
BTP	Business Transaction Protocol
CDR	Common Data Representation
CCI	Common Client Interface
CGI	Common Gateway Interface
CORBA	Common Object Request Broker Architecture
CPA	Collaboration Protocol Agreement
DCE	Distributed Computing Environment
DCOM	Distributed Component Object Model
DISCO	Discovery of Web Services
DOM	Document Object Model
DR	Network Data Representation
DTD	Document Type Definition
EAI	Enterprise Application Integration
ebXML	Electronic business XML
EDI	Electronic Data Interchange
ERP	Enterprise Resource Planning
FTP	File Transfer Protocol
HTML	Hypertext Markup Language
HTTP	Hypertext Transport Protocol

HTTPs	Secure HTTP
IIOP	Internet Inter-ORB Protocol
IOR	Interoperable Object Reference (IOR)
IOTP	Internet Open Trade Protocol
J2EE	Java 2 Platform, Enterprise Edition
JCA	J2EE Connector Architecture
JDBC	Java Database Connectivity
JMS	Java Message Services
JSP	Java Server Pages
JVM	Java Virtual Machines
MIME	Multipurpose Internet Mail Extensions
NASSL	Network Accessible Service Specification Language
OASIS	Organization for the Advancement of Structured Information Standards
ORB	Object Request Broker
QoS	Quality of Service
RDF	Resource Description Framework
RMI	Remote Method Invocation
RMP	Remote Method Protocol
RPC	Remote Procedure Call
SAML	Security Assertion Markup Language
SAX	Simple API for XML
SDL	Service Description Language
SCL	SOAP Contract Language
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
SOAP-DSIG	SOAP Digital Signatures
SQL	Structured Query Language
SSL	Secure Sockets Layer
TCP/IP	Transmission Control Protocol/Internet Protocol
UML	Unified Modeling Language
URI	Universal Resource Identifier
W3C	World Wide Web Consortium

WIDL	Web Interface Definition Language
WSCI	Web Service Choreography Interface
WS-COO	Web Services Coordination
WSDL	Web Services Description Language
WSFL	Web Services Flow Language
WS-I	Web Services Interoperability Organization
WSIL	Web Services Inspection Language
WS-TXN	Web Services Transactions
UDDI	Universal Description, Discovery, Integration.
XLink	XML Linking Language
XSL	XML Stylesheet Language
XSLT	XSL Transformations
XML	Extensible Markup Language
XMLP	XML Protocol

1. THESIS INTRODUCTION

1.1. Motivation

Currently the Web services landscape is in an evolving state with core specifications almost mature, whereas more specific standards for complex business requirements have not been finalized. The principal motive underlying this thesis is to highlight the potential of Web service solutions as an enabling distributed computing technology by studying the role, standards and deficiencies of Web service technologies. Another central objective is to provide a clear insight into Web Service Orchestration. The final goal is to produce a sample Web service scenario for the evaluation of Web service adoption.

The thesis study sets off with a state-of-the-art study on Web services technology and its radical changes on application innovation and development. Existing Web service standards are outlined, paying attention to their interrelationships and relative importance. Based on the investigation of Web service's recent shortcomings, development activities in the field of Web Service Orchestration are about to flourish in the upcoming days. Due to this fact, a major part of the study is devoted to taking different viewpoints on existing business process languages such as BPEL4WS, WSCI and BPML so as to produce a comparative study.

As a general scheme for Web service adoption does not exist, the work is concluded with a business scenario section for the evaluation of Web service development methodology. A financial Web service use case is used to investigate the feasibility of the Web services paradigm for application development. Due to evolving Web service standards, the possibility of incorporating future improvements and extensions is also taken into consideration.

1.2. Organization – Thesis Roadmap

Figure 1.1 illustrates the structure of the thesis and the relations between the different chapters.

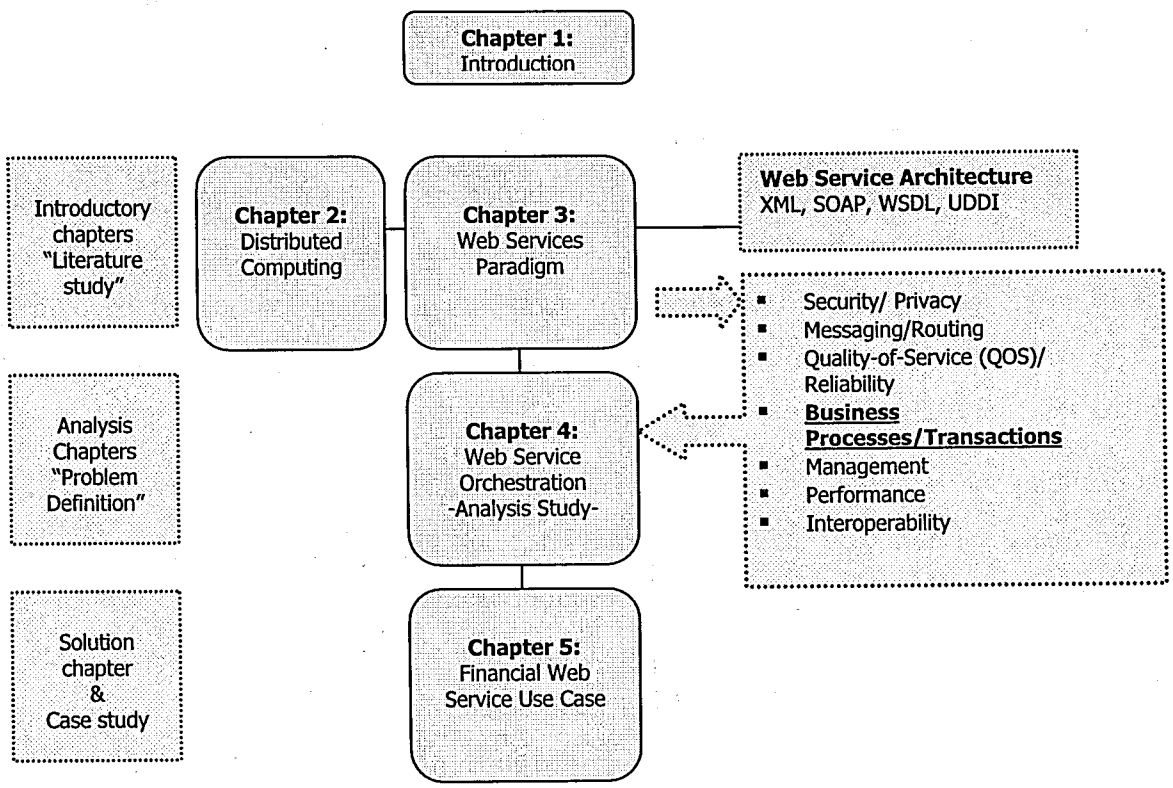

Figure 1.1. Thesis roadmap

2. DISTRIBUTED COMPUTING TECHNOLOGIES

2.1. General Introduction to Web Services

Web services are loosely coupled applications based on existing open-source Internet technologies to present well-documented, universally accessible interfaces. A variety of applications can be published as Web services such as stock quotes, weather reports, credit authorization, credit checking and loan approval, fulfillment and logistics, materials procurement, etc. within enterprise boundaries or across distributed platforms. A Web service can be accessed by a user through a desktop or mobile browser; however it can also be invoked by another application without any user intervention. Moreover, multiple services can interact to form a new composite Web service.

Main Web service standards providing message communication, service publication and discovery are Simple Object Access Protocol (SOAP), Web Service Description Language (WSDL) and Universal Description, Discovery, Integration (UDDI), which are all standards based on the Extensible Markup Language (XML). Web service descriptions are formed with WSDL, an XML-based interface language. A WSDL document of a service includes all the information required to locate and access a service. Such a service description is abstract and as a result a service can be used independently of its implementation hardware or software platform or its programming language. Therefore Web service technologies facilitate the development of loosely coupled, component-oriented, cross-technology applications (Kreger, 2001).

There are several key features differentiating Web services from other distributed computing technologies (Zakheim, 2002):

- Web services provide interoperability due to language, platform and vendor-neutrality.
- They are based on open Internet standards.

- They are ubiquitous as there is a broad agreement in industry to support Web service technologies and standards.
- They create a service-oriented (integration) architecture.

Joining all these characteristics together, it might become possible to develop completely reusable, universally accessible software on already existing distributed systems without making major changes for integration. Conclusively, Web services are a promising technology to solve the problem of enterprise application integration.

2.1.1. Definitions of Web Services

The industry agrees upon the fact that Web services reflect a profound advancement in distributed computing technology; however a single Web service definition does not exist.

Various ***Web service definitions*** have been listed down in the following paragraphs:

- A neutral description of Web services provided by (Graham *et al.*, 2002):

"A Web service is a platform and implementation independent software component that can be:

Described using a service description language
Published to a registry of services
Discovered through a standard mechanism (at runtime or design time)
Invoked through a declared API, usually over a network
Composed with other services"

- A description of Web services provided by (Kreger, 2001):

"A Web service is an interface that describes a collection of operations that are network-accessible through standardized XML messaging".

- Descriptions of Web services provided by Eric Newcomer (Newcomer, 2002):

  "Web services are Extensible Markup Language (XML) applications mapped to programs, objects, or databases or to comprehensive business functions."
  "...Web services provide the standard glue connecting diverse pieces of software."

- A description of Web services provided by Ethan Cerami (Cerami, 2002):

  "A Web service is any piece of software that makes itself available over the Internet and uses a standardized XML messaging system."

- A generic definition by the Web Services Interoperability Organization (WS-I):

  "WS-I views Web Services as having a general purpose architecture with inherent interoperability that supports B2B scenarios, but extends far beyond in scope and functionality".

- A definition given by WebServices.org:

  "Web Services are encapsulated, loosely coupled contracted functions offered via standard protocols" where:
  "Encapsulated" means the implementation of the function is never seen from the outside.
  "Loosely coupled" means changing the implementation of one function does not require change of the invoking function.
  "Contracted" means there are publicly available descriptions of the function's behavior, how to bind to the function as well as its input and output parameters.

- We decided that the W3C Web Services Architecture Group Web service definition (W3C, 2003) is the most applicable one in practical terms:

  "*A **Web service** is a software application identified by a URI, whose interfaces and binding are capable of being defined, described, and discovered by XML artifacts,*

and supports direct interactions with other software applications using XML-based messages via Internet-based protocols."

2.1.2. Dynamic E-business

If the role of Web services within business models is specified, they can be categorized as dynamic e-business. A simple definition of ***dynamic e-business*** according to IBM (Gisolfi, 2001a) would be: "*The next generation of e-business focusing on the integration and infrastructure complexities of B2B by leveraging the benefits of Internet standards and common infrastructure to produce optimal efficiencies for intra- and inter-enterprise computing.*"

There are some principles proposed for realizing the goals of dynamic e-business:

- Integration between software resources should be loosely coupled.
- Service interfaces for software resources should be universally published and made accessible.
- Program-to-program messaging must be compliant with open Internet standards.
- Applications can be constructed by stitching together core business processes with outsourced software components/resources.
- An increase in the availability of granular software resources should improve the flexibility and personalization of business processes.
- Reusable outsourced software resources should provide cost and/or productivity efficiencies to service consumers.
- Software can be sold as a service.

2.1.3. Overview of Service-oriented Architecture (SOA)

The ***Service Oriented Architecture (SOA)*** is the founding framework for the Web services environment. Computation is done in a service-oriented fashion and the key point is not the implementation but the service itself. Platform independence, code reusability and possibility of creating processes by functions compositions are some of the advantages

that a service-oriented architecture offers. The Web Services model is based upon the interactions between three roles in the SOA architecture: *service provider, service registry and service requestor*. The interactions involve *publish, find and bind* operations. A service provider creates a service, publishes its interface and *registers* the service with a service broker. A service requestor queries the service broker for a specific service. The service broker offers directive information for service inquiries. A service requestor binds to the service based on the underlying interface description, when a compatible service has been found. Details about the roles associated with Web services and their interactions are outlined in the next chapter.

2.1.4. Web Services Stack Architecture

The Web service architecture can be examined as a protocol stack initially with four basic layers: *Transport, XML messaging, service description, service discovery*. The current web service protocol stack is shown in Figure 2.1 (Cerami, 2002).

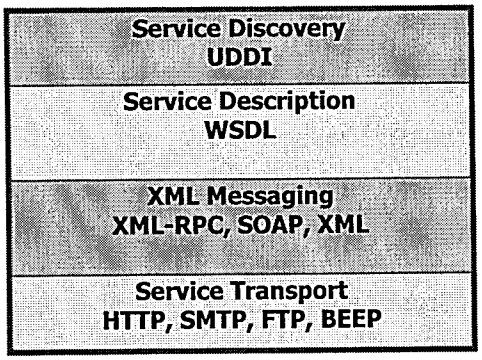

Figure 2.1. Web Service Protocol Stack

The *service transport layer* provides support for network transport protocols so that service messages can be exchanged across applications. The *XML messaging layer* is responsible for message encoding operations in a common XML format. The *service description layer* handles service descriptions based on WSDL. The *service discovery format* includes common registry features such as publishing, searching and invoking a service.

Various vendors work on specifications to create additional stacks for more complicated service requirements like Quality-of-Service (QoS), security, management, reliability, availability, etc. which constitute specific requirements of business applications in e-business services.

2.1.5. Enabling Technologies

Dynamic e-business is based on emerging web technologies that rely on open Internet standards: XML, WSDL, SOAP, UDDI.

XML represents a text-based markup language and a family of related specifications published and maintained by the World Wide Web Consortium (W3C). Unlike HTML, which uses tags for describing presentation and data, XML strictly serves as a base language for the definition of portable structured data. The XML language can be used for defining data description languages, interchange formats, validity checking and messaging protocols. The XML specification can be found at (W3C, 2000).

WSDL is an XML-based vocabulary that acts as the interface definition language for Web services. It can be used to define Web services interfaces, data and message types, interaction patterns, and protocol mappings. WSDL is used to describe request and response messages for remote method invocations (RMI) that are exchanged between the service provider and requestor. Message descriptions are abstract. Location and binding details, network protocol and message format definitions are specified during service interaction. Therefore WSDL is described as being platform and language neutral. The WSDL specification can be found at (W3C, 2002b). Except WSDL there are two other description languages, Resource Description Framework (RDF) and DARPA Agent Markup Language (DAML). These languages have a more complex structure, however they have no commercial support yet.

SOAP is an XML-based lightweight protocol for decentralized, cross-platform distributed computing. It provides document-centric message communication and remote

procedure calls, in an object-oriented programming fashion using XML, between service requestor and provider. The SOAP specification can be found at (W3C, 2002a).

UDDI is a registry service for Web services and acts like a search engine with the additional feature to register the provider's own service. The UDDI specification is made up of several SOAP APIs that provide the implementation of a service broker. Message communication is based on SOAP. UDDI provides basic functions and information for publishing, finding and binding to a provided service and information. Detailed information of UDDI is available at (Uddi.org, 2001).

2.2. Brief History of Distributed Computing Models

2.2.1. Distributed Computing Technologies

Cross platform support and integration have always been key issues in computing. Early initiatives in distributed computing were much more concerned with (Ogbuji, 2002) integrated data centers, whereas interoperability of heterogonous environments was left out of scope. Electronic Data Interchange (EDI) is one example for these initiatives. During the same period with EDI, minicomputers were being used for data communication across multiple machines. But due to the advent of PCs, minicomputers did not gain widespread interest.

2.2.2. Computer Networking

At the end of the 1980's, computer networking became a fundamental strategy for IT solutions. Instead of centralized data storage and services, much more companies decided to scale their systems on several centers and platforms and first a two-tier than a three-tier architecture was adopted. Since workload was distributed across multiple systems, bottlenecks were minimized. As a result, system integration at the network layer came into play. The need for a middleware layer that sits on top of the host's operating system and networking services and offers available services in a distributed fashion to clients became obvious. By that time, the distributed Unix Network File System (NFS) was developed by

Sun Microsystems. Microsoft's Distributed Computing Environment (DCE) model was another effort to standardize remote procedure call (RPC) technologies that were used for communication between distributed applications across multiple systems. This procedural programming model is based on messaging technologies combined with request/response design. DCE did not receive industry support, as it was superseded by another programming methodology that meanwhile emerged: Object-oriented programming.

With the introduction of object-oriented programming, many companies started to choose network-computing models based on object technology due its promised support for code reusability, modularity, etc.

The evolution of software programming methods with regard to distributed computing can be grouped into the following categories: Modular programming, libraries, dynamic libraries and object based communication protocols.

Modular programming is simply reduction of complexity. A programmer writes a function or procedure, which then can be reused within any other application coded in the same development environment.

Libraries are made up of files containing compiled and optimized function codes. A developer only has to link these files and can than access its functions. The drawback of using libraries is that actual linking occurs during program compilation. Therefore, updating the application due to library changes requires recompilation. Dynamic link libraries (DLL) offer an alternative by simplifying the update process. Static DLL's are included to an application at compile-time, however recompiling and redistributing the DLL is sufficient to make these changes available for applications at runtime. This is also a preferable method for memory related optimizations.

To turn to emerging technologies in the 90's; the evolution of new communication protocol models also started under the spell of object technologies and resulted in a protocol category based on Object RPC (ORPC), which includes Common Object Request Broker Architecture (CORBA), Component Object Model (COM), Distributed Common

Object Model (DCOM) and COM+, Enterprise Java Beans (EJB) and Remote Method Invocation (RMI).

2.2.3. Common Object Request Broker Architecture (CORBA)

The Object Management Group (OMG) introduced ***CORBA*** in 1991 as a language-neutral model for object-level communication and interoperability in distributed computing environments. The main definitions of CORBA include the Interface Definition Language (IDL), the Application Programming Interface (API) and Object Request Broker (ORB). CORBA uses Internet Inter-ORB Protocol (IIOP) to communicate between different systems.

IDL is used to define high level of abstraction of the language mappings and objects between CORBA and other programming languages. Therefore clients and servers can be written in any programming language. A compiler handles the mappings of an IDL file to a specific programming language. An ORB can be described as a middleware used for the communication between objects, clients, and servers in requestor-provider actions within a distributed system (Gisolfi, 2001). Due to incompatible ORB implementations, interoperability problems occurred and CORBA 2.0 was released at the end of 1994 to overcome these obstacles.

CORBA permits only one interface implementation per object. An Interoperable Object Reference (IOR) represents the object reference. This representation stores the address of a CORBA object in a portable format to resolve its reference. Payload parameter format is Common Data Representation (CDR).

2.2.4. Component Object Model (COM)

As an alternative for CORBA, Microsoft released COM and DCOM. The language-neutral COM specification combines features like code reusability, reduction of memory usage, simplified application updates, platform independence etc for object communication through COM interfaces.

2.2.5. Distributed Component Object Model (DCOM)

COM technology was extended into ***DCOM***, which is a way of using objects to communicate over a network with various network protocols like TCP/IP and HTTP.

The DCOM model enables clients to use local and remote components transparently through a layer above the DCE RPC mechanism and which interacts with the COM runtime services. Interfaces for clients are written in IDL, which is similar to C++, then compiled and registered in the system registry. The Object Remote Procedure Call (ORPC) is DCOM's remote call protocol. Object reference representation for identification and access is done trough OBJREF's. Payload parameter format is Network Data Representation (DR). Support for type libraries and multiple interfaces per object exist. These are files that describe the remote object and which can be queried by interfaces provided by the

DCOM requires specific configurations; especially configuration settings made to establish access in object-communication across firewall-boundaries are complicated. This is due to the fact that firewalls initially only provide connection though port 80, but DCOM makes use of different port numbers. COM finally evolved into its latest version named COM+.

2.2.6. EJB/RMI/IIOP

Enterprise JavaBeans (EJB) is a platform independent, but JAVA dependent specification released by Sun Microsystems. The communication protocol for EJB is Remote Method Invocation (RMI) over IIOP.

RMI is a programming model for developing distributed Java-to-Java applications, in which the methods of remote Java objects can be invoked from other Java virtual machines (JVM). In RMI, references for remote objects are obtained through lookup in a naming service or by receiving references as arguments or return values. The protocol used for the

communication is the Java Remote Method Protocol (JRMP). An abstract language like IDL is not used to describe the remote server object.

A timescale in Figure 2.2 displays the evolution and lifecycle of significant distributed computing technologies.

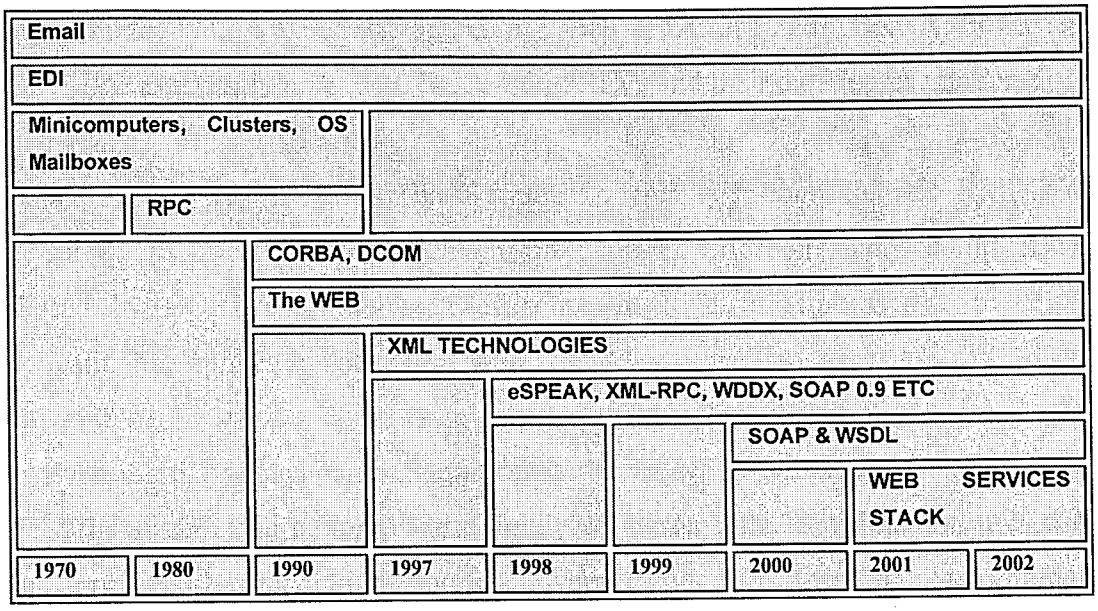

Figure 2.2. Evolution of distributed computing technologies

2.2.7. Disadvantages of Former Technologies

As these solutions not build upon open standards, they do not fulfill the assertion of being completely vendor independent and do not offer full interoperability for complex operations in distributed systems. Furthermore, there is the difficulty of adopting protocol based messaging and data formats within an existing system. Another side effect of these non-standard solutions is that there are rare opportunities to integrate enhancements that are produced by different sources.

Although CORBA is more widely implemented than COM technologies, the incompatibility of ORB implementations by competing vendors is a major problem. Moreover, implementation of distributed CORBA applications is fairly complicated,

because to enable IDL to be translated into various languages, it has to limit itself to concepts that are found in all the supported languages, thus it represents a least common denominator (Gunzer, 2002).

Even though DCOM is vendor independent, in some cases compatibility problems can occur, when companies prefer to use several distributed models that do not always use the same methods to provide interoperability such as DCOM and CORBA. For example, CORBA IORs cannot interact with DCOM OBJREFs and their data representation differs.

With RMI, only distributed Java applications can get connected, so RMI is not a universal solution as a communication protocol. In addition many complex issues addressed by CORBA are left out of scope in RMI to simplify the model, thus complicating the application development phase.

Even if a distributed computing infrastructure is set up based on CORBA or DCOM models, the system will merely have a tightly coupled nature. Developers follow a hard-wiring approach; they have to tell cooperating applications how these should communicate and share data with each other. In the end, applications require complicated design considerations, long-term management and controls. Connections are established on a point-to-point basis, which might only be considered as appropriate for intranet applications behind the firewall. Tight coupling is only beneficial for some reasons; the location and behavior of cooperating applications is known. Therefore specific issues like QoS, security, data integrity etc. can be handled more safely.

2.2.8. Basic Requirements of a Distributed Computing Model

The following list provides a generalized overview of the features in an idealized distributed computing model:

- Vendor, platform and language independent standardized specifications.
- Standards-based. Existing standardized IT and networking infrastructure need to be supported.

- Despite addressing integration as its key goal, ease of development and adoption has also to be taken into consideration.
- Support for internal and external data communication and exchange is necessary, so that the distributed environment term is not only limited to the systems behind the company firewall.
- The architecture should be loosely coupled, but still offer features like scalability, management, performance monitoring, QoS and security.
- The model should also enable sophisticated business processes, transactions and workflow besides basic messaging operations.
- Resilience to environment changes and seamless interoperability.
- A combination of a simplified yet interoperable technology.
- Globalization.

2.2.9. Web Services Come Into Play

HP announced e-Speak, the first commercial Web service technology, in 1999. e-Speak described network systems as e-services and exploited XML and HTTP for data exchange and transfer.

At the same time, XML based messaging formats emerged. The Userland community developed XML-RPC, a royalty-free specification for remote method calls in distributed systems.

The EDI industry was also interested in Internet and XML based protocols and sample developments can be described as early Web service examples for business-to-business transaction. In 1999, these EDI/XML models triggered an effort for e-business XML (ebXML), a joint project of OASIS, an SGML/XML pioneer, and the United Nations Center for Trade Facilitation and Electronic Business (UN/CEFACT), a key organization in the development of traditional EDI.

The Simple Object Access Protocol (SOAP) was aimed to become a specification for structured exchange of XML documents. DevelopMentor and Microsoft developed the

draft specification of Simple Object Access Protocol (SOAP). Its first public release took place in 1999 and it gained support from many commercial vendors and developer communities. It is a communication protocol like XML-RPC, however with more complex features to overcome its shortcomings. Since it has been submitted to the Internet Engineering Task Force (IETF) and is now being ratified by the W3C, it has become an open-standard for distributed computing. With the growing support for SOAP, the need for a standardized specification describing Web services was realized and many companies started to work on specifications, which are listed in Table 2.1.

Table 2.1. Web Service Description Languages

Company	Specification
WebMethods	▪ Web Interface Definition Language (WIDL)
Microsoft	▪ Service Description Language (SDL) ▪ SOAP Contract Language (SCL) ▪ Discovery of Web Services (DISCO)
IBM	▪ Network Accessible Service Specification Language (NASSL) ▪ Advertisement and Discovery of Services (ADS)

These standardization efforts ended with the release of Web Services Description Language (WSDL), originally by IBM, Microsoft and Ariba. This progress continued with the release of the Universal Description, Discovery and Integration (UDDI) system used for discovery of Web services through categorized directories. UDDI was formed by convergence of specifications like Discovery of Web Services (DISCO) and Advertisement and Discovery of Services (ADS) by a consortium of 36 companies, which soon attracted many more supporters.

As the following chapters outline, core Web Service specifications are almost standardized and constitute the fundamental protocols within the Web Services Architecture. Yet companies now work on several specifications to address issues like security, performance, Quality-of-Service (QoS), business processes, service level agreements (SLA) etc. to fill up the stack. On top of the basic standards such as SOAP and

WSDL, many higher-level standards are proposed and many business-specific and sector-specific standards are arising.

2.2.10. What Makes Web Services Different?

Web services are a distributed computing architecture designed and specified to foster cross-platform program-to-program communications (Bloor Research, 2002). Based on the overview section about distributed computing, Web services emerge not as a revolution, however an evolution among the existing technologies within this field. With a technological perspective, the Web Services concept is truly not new, as it offers the same functionalities of CORBA or DCOM like modularity, encapsulation and loose coupling. However, what Web Services turns into a breakthrough solution is the simplicity of integration due to open Internet standards and that XML plays an important role in the definition, implementation, and execution of Web services. The major differences of distributed computing architectures are displayed in Table 2.2.

Web services exploit already widely used Internet protocols, which makes them a promising alternative in distributed environments. E.g., Data communication uses the Hyper Text Transfer Protocol (HTTP), which makes Web services firewall friendly and payload agnostic (Gisolfi, 2001). In addition, object identification can be done trough URL's, which all Internet users are familiar with. To use CORBA ORB frameworks a certain fee has to be paid, whereas HTTP/SOAP is a free of charge middleware. The encoding parameter schema for Web services is XML instead of DR and CDR data representation formats.

SOAP combines XML and RPC for transparent data transmission and method invocation and makes use of existing Internet protocols like HTTP, which turns it also into a firewall-friendly protocol. In general, with these features SOAP can claim to become the essential -platform and language neutral- distributed computing solution. Companies already work on interoperable SOAP implementations and create application extensions.

Table 2.2. Client/Server components and implementation attributes in different RPC architectures

RPC Architecture	CORBA	DCOM	RMI/IIOP	Web Services/SOAP
Client Stub	Client Stub	Proxy	-	Service proxy
Server Stub	Skeleton	Stub	-	Service implementation template
Implementation Attributes	**CORBA**	**DCOM**	**RMI/IIOP**	**Web Services**
Endpoint Naming	IOR	OBJREF	OBJREF	URL
Interfaces/Object	Single	Multiple	Multiple	Multiple WSDL
Payload Parameter Value Format	DR	CDR	CDR	XML
Format	Binary	Binary	Binary	XML
Firewall friendly	-	-	-	-
Platform	Unix	Windows	Independent	Independent
Programming language	Independent	Independent	JAVA	-
Access through trusted domains	-	-	-	Yes

Open standard specifications and Internet protocols make Web services modular, self-describing and self-contained applications that are accessible over the Internet. Distributed applications developed in different languages on different platforms can easily be integrated and executed. As core standards reached a certain level of maturity, developed applications can support future enhancements and are resilient to changes, which is a precondition for seamless interoperability.

Another aspect in comparing distributed computing technologies is tight-loose coupling. CORBA, J2EE, and DCOM, all of which are based on RPC concepts, transfer message mappings in a tight-coupled manner within the interface, whereas in Web services XML parsers are used for message mappings on the client-side. Due to this fact, Web services do not resemble remote procedure call invocations or software components. Web services have a loosely coupled architecture, which simplifies application development and maintenance in distributed systems. It goes without saying that a higher level of flexibility and interoperability can be reached through a loosely coupled architecture for application-

to-application communication across different platforms. However loosely coupled Web services in their current state may need additional integrated solutions to support performance, security, transactions and similar mission-critical features to a certain extent.

3. WEB SERVICES PARADIGM

3.1. Web Service Architecture

3.1.1. Service-oriented Architecture (SOA)

The conceptual representation of Web services is based on the service-oriented architecture (SOA). The term "service-oriented architecture" was firstly proposed by the analyst firm Gartner Group (Zakheim, 2002). In an SOA, well-defined interfaces provide access to organization-wide services. Moreover, business logic and data can be shared and reuse of existing services through alternative communication channels with different configurations.

SOA characteristics can be summarized by four chief properties: *Distributed, loosely coupled, standards based and process centric* (Sleeper and Robins, 2002). Application deployment is performed across multiple even remote platforms with flexible coordination controls. The entire design, development and design phase is based on vendor-independent standards. As mentioned before, the main player within the Web Service Architecture is the service. With regard to a process-centric perspective, workflows and business processes are comprised of multiple steps, which are actually services. In this context, a service can be described as a software module deployed on network accessible platforms provided by the service provider (Kreger, 2001). Services can be invoked by service requestors or interact with other services. The service description contains information about data types, operations, bindings, network locations, categorization related with a service interface. Publishing a service description to a service requestor or to a service registry makes it available for invocation.

3.1.2. Roles, Operations and Artifacts in the Web Services Model

There are several roles in the SOA model. The Web service *actors, objects and operations* can be seen in Figure 3.1:

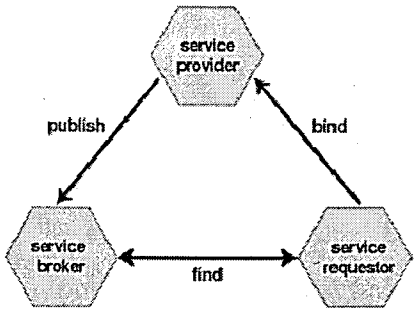

Figure 3.1. SOA participants

- **Service Provider:** This network node creates a service interface, describing a service it provides, and publishes it within a service registry. Service requestors access and invoke the services of this distributed object. The service provider entity almost acts like a server in a client-server architecture. From a business perspective, this is the owner of the service.
- **Service Requestor:** This network node discovers and invokes other software services to provide a business solution. The requestor role can be a business application component, such as a program or another Web service without a user interface, performing remote procedure calls to a service provider. A provider can reside locally within an intranet, or remotely over the Internet. Due to the conceptual nature of SOA, networking protocols, transport protocol, and security details are left to the specific implementation. The service requestor entity almost acts like a client in client-server architecture.
- **Service Broker:** This network node acts as a searchable registry, such as yellow pages, where service providers can publish their service interfaces. The service broker advertises Web service descriptions and service requestors search the registry's service description collection. As soon as a requestor specifies a certain service, a direct interaction between the service requestor and provider takes place. If the service requestor and provider have a static binding, the service registry is an optional role in the architecture, since provider and requestor can directly exchange information. Service requestors can also obtain a service description from other

sources such as a local file, FTP site, Web site, Advertisement and Discovery of Services (ADS) or Discovery of Web Services (DISCO).

Basic SOA operations are: *Publish, find, and bind.* These operations can occur as single or iterative actions. A service provider publishes a service to a service broker to make it accessible. The place of publication can vary. A service requester finds a required service using lookup operations of a service broker. While a find operation at design time can be used for client-side application development, a find operation at runtime is the initial step for a bind operation. A service requester binds to a service of a certain service provider, if a compatible service based on the discovery criteria has been found. Finally, the bind operation provides information for the service requestor to initiate actual service invocation.

3.1.3. The Web Services Stack

The ***Web Services Programming Stack*** includes standardized protocols and application programming interfaces (APIs) for utilizing Web services. Figure 3.2 outlines all layers contained in the conceptual Web services stack:

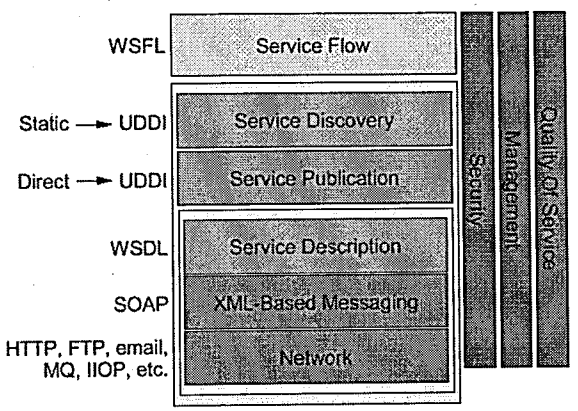

Figure 3.2. Conceptual Web services stack

The first three layers - Network, XML-Based Messaging and Service Description are required for interoperability among Web services. The *base layer* deals with network related issues for Web services. The *network layer* supports several network protocols such

as HTTP, Internet Inter-ORB Protocol (IIOP) or IBM MQSeries. The XML-based *messaging layer* is placed above the network layer and provides Web service and client communication by utilizing the Simple Object Access Protocol (SOAP). The *service description layer* is based on the Web Services Description Language (WSDL). WSDL specifies an XML-vocabulary for the description of Web services programming interfaces and locations. The next three layers - Service Publication, Discovery and Flow are optional. Through *service publication*, service requestors can collect information about a service and service access and execution becomes possible. Publishing a service can be through various ways, such as e-mail or storing the WSDL document of a service on a UDDI registry. *Service discovery* describes any action performed by a service requestor to access a Web services WSDL document. The management and composition of Web services is handled in the *service flow layer*. Flow definitions are useful for specifying process flows within a workflow automatically.

The vertical stack layers Quality-of-Service (QoS), Security and Management show infrastructure services such as reliability, availability, manageability, security, etc. which constitute specific requirements of business applications in e-business services.

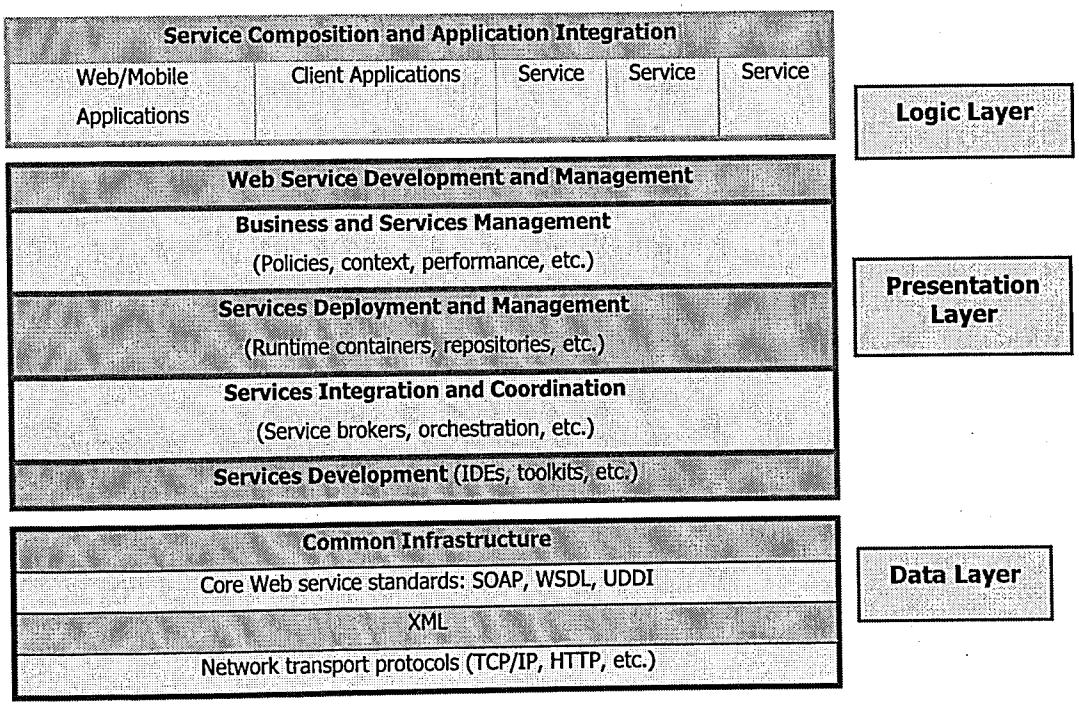

Figure 3.3. Extended Web services stack

Core Web service standards such as SOAP, WSDL and UDDI cannot provide complex requirements for sophisticated business applications. Therefore existing Web services are limited with static features. To provide a solid computing infrastructure for e-business, principal issues like security, reliable messaging, quality of service, performance, management, service composition, business process and transaction, workflow have to be supported. With evolving Web service standards and ongoing specification efforts, higher-level functionalities will become available. An example of the extended Web Service Architecture stack is illustrated in Figure 3.3 (Sleeper, 2002).

3.2. Principal Web Service Standards

At the heart of Web service standards are SOAP, WSDL, and UDDI, which provide a language-independent, platform-neutral and flexible distributed computing infrastructure for "plug and play" applications. Before giving detailed information about core Web service standards, outlining interaction flow step-by-step will provide a general overview:

- The service provider publishes a service in the UDDI Registry.
- The service requestor 'finds' a specific service in the UDDI Registry.
- UDDI Registry provides a pointer to the service access-point (WSDL).
- The service requestor uses the 'WSDL' document to create proxy object.
- The proxy object can be used to access and invoke the specified service remotely.

The interaction of Web service standards is displayed in Figure 3.4 (Ajay, 2002):

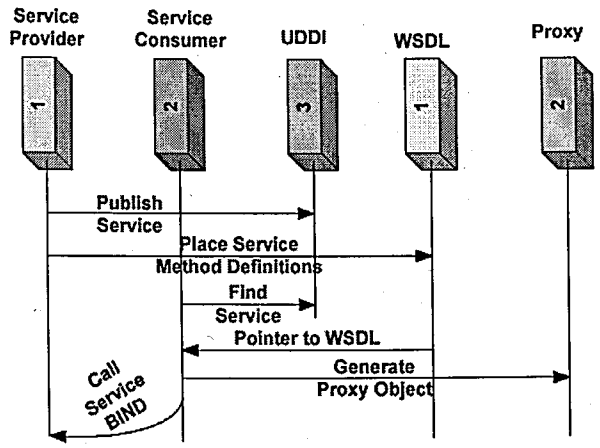

Figure 3.4. Interaction of Web service standards

3.2.1. The Extensible Markup Language (XML)

Even though the Hypertext Markup Language (HTML) is a de facto standard for static content creation, it offers only limited capabilities for the support of dynamic content creation and management. Today we see how the Web becomes a software-platform, in which data is not only a means of presentation but also has associated meaning, so that content can be dynamically created and evaluated. To overcome the limitations of HTML, the Extensible Markup Language (XML) has been developed. Unlike HTML, which uses tags for describing presentation and data, XML strictly serves as a base language for the definition of portable structured data so that associated schemas can validate data and describe other attributes and qualities of the data.

XML represents a text-based markup language and a family of related specifications published and maintained by the World Wide Web Consortium (W3C). The XML language can be used for defining data description languages, interchange formats, validity checking and messaging protocols. Definitely XML can be nominated as the foundation of Web service standards and naturally in the context of Web services it is more than only a markup language for formatting data. Core Web service standards such as SOAP and WSDL are XML-based languages and their extensibility mechanism relies on XML. New extensions or specifications derived from existing standards will naturally also become

XML-based languages. A key characteristic of Web services is that they are self-describing. This feature is basically provided by XML.

The family of XML standards consists of a data markup language, various content models, a linking model, the XML namespace model, and various transformation mechanisms. The following list includes the most important specifications, which are also extensively used in Web services:

- **XML v1.0.** The rules for defining elements, attributes, and tags enclosed within a document root element, providing an abstract data model and serialization format.
- **XML Schema.** XML documents that define the data types, content, structure, and allowed elements in an associated XML document; also used to describe semantic-processing instructions associated with document elements.
- **XML Namespaces.** The uniquely qualified names for XML document elements and applications.
- **XML Information Set.** A consistent, abstract representation of the parts of an XML document.
- **XPointer.** A pointer to a specific part of a document; XPath, expressions for searching XML documents; and XLink, for searching mulitple XML documents.
- **Extensible Stylesheet Language Transformations (XSLT).** Transformation for XML documents into other XML document formats or for exporting into non-XML formats.
- **DOM (Document Object Module) and SAX (Simple API for XML).** Programming libraries and models for parsing XML documents, either by creating an entire tree to be traversed or by reading and responding to XML elements one by one.

3.2.2. The Simple Object Access Protocol (SOAP)

The SOAP specification was co-authored by Microsoft, IBM, Lotus, UserLand, and DevelopMentor. The specification called SOAP 1.1 was submitted to the World Wide Web Consortium (W3C) to initiate a standardization effort. The W3C XML Protocol Activity

Group has an ongoing development process for defining a standard XML Protocol called SOAP 1.2. SOAP Version 1.2 is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment (W3C, 2002a). The protocol is completely neutral with respect to operating system, programming language, or distributed computing platform.

SOAP extends the servlet/cgi request processing on a Web server by standardizing request/response messages with the use of XML. SOAP is fundamentally a stateless, one-way message exchange paradigm, but applications can create more complex interaction patterns (e.g., request/response, request/multiple responses, etc.) by combining such one-way exchanges with features provided by an underlying protocol and/or application-specific information (W3C, 2002a). Since the major design goals for SOAP are simplicity and extensibility, SOAP omits features from the messaging framework such as "reliability", "security", "correlation", "routing", and the concept of message exchange patterns to be defined as extensions by other specifications (W3C, 2002a). Furthermore its extensible messaging framework, based on XML technologies, is independent of any particular programming model and other implementation specific semantics.

The one-way SOAP communication model ensures that a coherent message is transferred from sender to receiver, potentially including intermediaries that can process part of or add to the message unit. A SOAP message includes a SOAP envelope made up of a SOAP Header and a SOAP Body for Web services communication as shown in Figure 3.5 (W3C, 2002a). The envelope defines the start and the end of the message. The SOAP header provides an extensible mechanism for optional attributes to supply directive or control information about the message, either at an intermediary point or at the ultimate end point. These optional header attributes could be used to implement transactions, security, and reliability or payment mechanisms. The message body contains the XML payload of the SOAP message being sent. For a valid SOAP message only the envelope and the body are required, however there are several optional parts of SOAP such as Encoding, RPC interaction and Attachment.

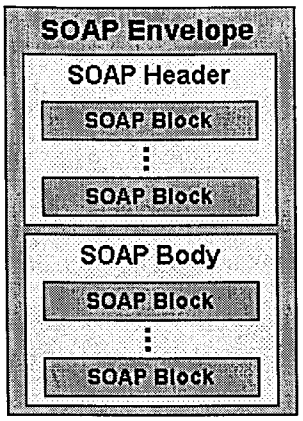

Figure 3.5. SOAP envelope

The following "CardValidator" example from "http://webservices.imacination.com /validate/Validate.jws" provides a basic overview of SOAP messaging. This example has been selected, as it resembles a simplified form of the "Payment Authorization" use case demonstrated in Chapter 5. The messaging has been captured with the SoapSpy tool contained within the WaspDeveloper Eclipse (Eclipse, 2003) plugin from Systinet (Systinet, 2002). The request and response messages include a single SOAP Envelope element, which in turn include a mandatory Body element. The main payload is the remote method name; for the request message the method name is "validateCard", which is tied to the "Validate" service listed in the URL mentioned above. The SOAP request method,parameters are listed as subelements: ccNumber – Credit card number, ccDate – Card expiry date. The request message is displayed in Table 3.1.

The SOAP response message "validateCardMethod" method relates to the "validateCard" request. "ValidateCardReturn" is the only response parameter; it returns a boolean value that determines if a card is valid or invalid. The overall response message is displayed in Table 3.2.

A SOAP message can be mapped to transport protocols by a special binding mechanism provided by the SOAP Transport Binding Framework. The SOAP 1.2 specification defines transport bindings for HTTP and the HTTP Extension Framework. There are SMTP, JMS and MQSeries bindings available.

Table 3.1. SOAP request message

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<e:Envelope xmlns:e="http://schemas.xmlsoap.org/soap/envelope/">
  <e:Body e:encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
  xmlns:d="http://www.w3.org/2001/XMLSchema"
  xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
      <n0:validateCard
      xmlns:n0="http://webservices.imacination.com/validate/Validate.jws">
       <ccNumber i:type="d:string">5321-7123-5712-3570</ccNumber>
            <ccDate i:type="d:string">11/2003</ccDate>
    </n0:validateCard>
  </e:Body>
</e:Envelope>
```

Table 3.2. SOAP response message

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <soapenv:Body>
          <ns1:validateCardResponse
          soapenv:encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
          xmlns:ns1="http://webservices.imacination.com/validate/Validate.jws">
  <validateCardReturn xsi:type="xsd:boolean">true</validateCardReturn>
          </ns1:validateCardResponse>
         </soapenv:Body>
</soapenv:Envelope>
```

SOAP Encoding Rules provide a serialization and binary encoding format for exchanging data and supporting abstract basic (scalar types: string, integer, enumeration, etc.) and complex data types (struct, array, etc.). The SOAP encoding rules are based on a simple type system derived from the W3C XML Schema Part 2: Datatypes Recommendation (W3C, 2001c). Basically data within SOAP message bodies are encoded as literals using XML. There is no default serialization mechanism to map application-defined data types to XML elements. To overcome this problem, users can apply SOAP encoding rules or their own serialization mechanism to transport encoded application-specific data. In a SOAP communication, the sender and receiver can decide on their own encoding rules through private agreement.

SOAP messaging provides support for loosely coupled and tightly coupled communication. In case of a loosely coupled communication, only the message format and access point URI of a service are known. The message sender has no details of the service implementation. The service provider determines how a service request is processed based on the received message content. As mentioned, SOAP also includes a programming convention for representing Remote Procedure Call interactions and responses as SOAP messages for a more tightly coupled communication scheme based on the SOAP RPC representation. If SOAP RPC is used, a SOAP request resembles a method call with zero or more parameters, whereas the SOAP response returns a return value and zero or more parameters. SOAP RPC requests and responses are marshaled into a "struct" datatype and passed in the SOAP body. Briefly, RPC interaction defines how to model RPC-style interactions with SOAP.

It is also possible to transport non-XML data, such as multimedia files, as attachments to a SOAP message with the SOAP with Attachments binding. Microsoft and Hewlett-Packard developed the related specification to define a SOAP 1.1 binding for MIME multipart/related messages (W3C, 2001a).

Recently there are various SOAP implementations, which mostly conform to the SOAP 1.1 specification. In order to prevent future interoperability problems, developers need to stick to the specification conventions.

3.2.3. Web Services Description Language (WSDL)

The Web Services Description Language (WSDL) is an XML schema format that defines an extensible framework for describing Web service interfaces based on an abstract model of what the service offers (W3C, 2002b). WSDL is built on top of standards like XML, XSD, SOAP and MIME. Since Web services are described to be "self-describing", as summarized in Chapter 2, there must be an interface and a publishing mechanism, which enables service invocation for service requestors, without knowing further details about the actual service implementation. By using WSDL, it is possible to separate the abstract service function descriptions from concrete service detail descriptions.

WSDL is a merged development of IBM's *Network Accessible Service Specification Language (NASSL)* and *Microsoft's Service Description Language (SDL)* and *Service Contract Language (SCL)* language. The WSDL 1.1 specification was submitted to W3C in March 2001 (W3C, 2001b). In spite of W3C's acknowledgement for the submission, no activity has been initiated to standardize the specification, yet WSDL is already widely implemented.

WSDL facilitates the communication between service requestor and provider on a common set of conventions to describe the operations a Web Service can perform, the formats of the messages that it can process, the protocols that it supports, and the access point of an instance of the Web Service. Like the other core standards of the Web services framework, WSDL is designed for use with both procedure-oriented and document-oriented interactions.

A WSDL document defines services as collections of network endpoints, or ports, and each port, defined abstractly as a port type, includes information about the set of operations supported by the service. In WSDL, the abstract service definition (of endpoints and messages) is separated from the concrete implementation details (network deployment or data format bindings). This allows reuse and supports a modular approach, providing the same abstract service via different channels.

The detailed document structure can be described by the *WSDL syntax, data type definitions, abstract definitions and service bindings*. The following categories list general features of the structure:

The structure of a WSDL document contains a set of element definitions that make up the **WSDL syntax**: *Types, messages, operations, port types, bindings, ports, and services.* All of these service descriptions can be placed within a single WSDL document or specified in separate XML document fragments and imported in various combinations to create a finalized Web services description. The major elements are explained in detail in the following paragraphs.

WSDL document elements are shown in Figure 3.6 (Newcomer, 2002). The elements are layered according to their levels of abstraction. As they are defined independently of the transport, the same service can use multiple transport protocols such as SOAP over HTTP or SOAP over JMS. The same separation idea applies to data type definitions so that they can be used by multiple services.

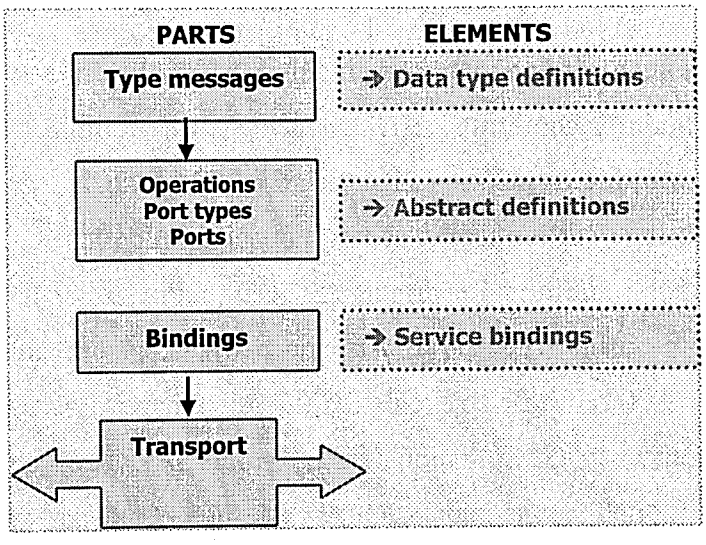

Figure 3.6. WSDL document elements

The "CardValidator" example outlined in the SOAP section service will be used to summarize the implementation of a WSDL document. Table 3.3 displays the overall structure of the example WSDL document. The plus signs are not used in the actual

coding; they are only placeholders for WSDL elements that contain sub-elements. Code related to each element is displayed in a separate table following element explanations.

Table 3.3. WSDL document example

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions
targetNamespace="http://webservices.imacination.com/validate/Validate.jws"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://webservices.imacination.com/validate/Validate.jws"
xmlns:intf="http://webservices.imacination.com/validate/Validate.jws"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
+ <wsdl:message name="validateCardResponse">
+ <wsdl:message name="validateNumberRequest">
+ <wsdl:message name="validateNumberResponse">
+ <wsdl:message name="validateCardRequest">
+ <wsdl:portType name="Validate">
+ <wsdl:binding name="ValidateSoapBinding" type="impl:Validate">
+ <wsdl:service name="ValidateService">
</wsdl:definitions>
```

Data Type Definitions determine the structure and the content of the messages and the data type definition layer includes type definitions. Although WSDL data type definitions are based on XML schemas, it is possible to substitute data type definition systems provided that communication parties agreed on the type definition system to be used beforehand. For example, COBRA Interface Definition Language (IDL) data types could be used instead of XML schema data types. To ensure the uniqueness of XML element names used in each major WSDL element definition, XML namespaces are used. Within a single WSDL document, overlapping XML namespaces are not allowed. Associated XML schemas are used for WSDL document validation.

The WSDL <types> element defines the data types that are used within messages. WSDL uses the datatyping system defined in the W3C XML Schema Part 2:Datatypes Recommendation (W3, 2001c) as its canonical type system, although any datatyping system may be used. The type system supports simple scalar types and complex types.

Schema types can be mapped to most programming language type systems. SOAP tools use the type information to encode and decode the data in SOAP messages.

Table 3.4 shows XML namespace usage and WSDL message elements. This code section displays how input and output parameters of the service request are coded within message elements with responding data type definitions.

Table 3.4. WSDL message elements

```
<wsdl:definitions
targetNamespace="http://webservices.imacination.com/validate/Validate.jws"
xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:apachesoap="http://xml.apache.org/xml-
soap" xmlns:impl="http://webservices.imacination.com/validate/Validate.jws"
xmlns:intf="http://webservices.imacination.com/validate/Validate.jws"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<wsdl:message name="validateCardResponse">
        <wsdl:part name="validateCardReturn" type="xsd:boolean" />
</wsdl:message>
<wsdl:message name="validateNumberRequest">
        <wsdl:part name="ccNumber" type="xsd:string" />
</wsdl:message>
<wsdl:message name="validateNumberResponse">
        <wsdl:part name="validateNumberReturn" type="xsd:boolean" />
</wsdl:message>
<wsdl:message name="validateCardRequest">
        <wsdl:part name="ccNumber" type="xsd:string" />
        <wsdl:part name="ccDate" type="xsd:string" />
</wsdl:message>
```

The **abstract definition layer** determines the operations performed on the message content based on the element definitions of data types, messages, and abstract operations, which are similar to interface definitions in CORBA or DCOM. The main elements within this layer are *operations, port types and ports*, which altogether form a message.

The WSDL <message> element defines the format of a message. Messages are used as input or output structures for operations. A message can consist of multiple logical parts, each of which is associated with a type. Messages can be defined for use with the

procedure-oriented interaction style, the document-processing interaction style, or both. When using the SOAP RPC programming model, each part represents a method parameter.

A port instantiates a port type and binding at a specific network address. A <portType> element defines a set of operations. The <operation> elements process a particular set of messages; they can be compared to methods in Object-oriented programming. Table 3.5 displays port type definitions for the "CardValidator" WSDL document. When using the SOAP RPC programming model, each operation represents a method. Through the abstractly separated layers, the same message can be used for multiple port types. Message definitions can also contain optional attributes due to the WSDL extensibility mechanism.

Table 3.5. WSDL port type elements

```
<wsdl:portType name="Validate">
<wsdl:operation name="validateCard" parameterOrder="ccNumber ccDate">
        <wsdl:input message="impl:validateCardRequest" name="validateCardRequest"/>
        <wsdl:output message="impl:validateCardResponse"
        name="validateCardResponse"/>
</wsdl:operation>
      <wsdl:operation name="validateNumber" parameterOrder="ccNumber">
        <wsdl:input message="impl:validateNumberRequest"
        name="validateNumberRequest"/>
        <wsdl:output message="impl:validateNumberResponse"
        name="validateNumberResponse" />
</wsdl:operation>
</wsdl:portType>
```

Service Bindings map the abstract messages and operations onto a data format and specific network transports, such as SOAP, which carry the message to its destination. The binding layer also provides extensibility components, which can be used to include information specific to SOAP and various other physical transport mappings. The WSDL specification offers support for SOAP one-way mappings for Simple Mail Transfer Protocol (SMTP), SOAP RPC mappings for HTTP, SOAP mappings to HTTP GET and POST, and a mapping for the Multipurpose Internet Messaging Extensions (MIME) multipart binding for SOAP.

A <binding> element maps the operations and messages defined in a port type to a concrete protocol and data format specification. A sample protocol and a sample data format specification could be respectively HTTP and the SOAP data encoding system. A <service> element defines a collection of related ports. A <port> element maps a binding to the location of an instance of the Web Service. For reusability and flexibility reasons, binding elements and service elements can be maintained in separate WSDL documents. Table 3.6 is the actual service binding code section of the example WSDL file.

The following steps cover up the complete description process:

- The type, message, and port type elements define a service in an abstract way, so they describe a service type.
- The binding element maps the service type to a specific protocol.
- The service element maps the service type and the binding to a specific instance of the service.

WSDL is designed as an extensible XML framework that can easily be adapted to multiple data type mappings, message type definitions, operations, and transports. The Internet Engineering Task Force (IETF) defined the connection-oriented Internet protocol Blocks Extensible Exchange Protocol (BEEP) as a protocol proposal for connection-oriented transport; BEEP especially addresses QoS shortcomings at the transport level. WSDL can easily be extended to map to the BEEP protocol, though recently companies prefer to stick to more traditional protocols like DCOM, CORBA or IIOP (Newcomer, 2002). In conclusion, due to its support for optional attributes and its extensibility mechanism, there is also the possibility, that different WSDL implementations raise interoperability and compatibility difficulties.

Table 3.6. WSDL service binding elements

```
<wsdl:binding name="ValidateSoapBinding" type="impl:Validate">
<wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
<wsdl:operation name="validateCard">
        <wsdlsoap:operation soapAction="" />
        <wsdl:input name="validateCardRequest">
                <wsdlsoap:body
                encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
                namespace="http://webservices.imacination.com/validate/Validate.jws"
                use="encoded"/>
 </wsdl:input>
        <wsdl:output name="validateCardResponse">
                <wsdlsoap:body
                encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
                namespace="http://webservices.imacination.com/validate/Validate.jws"
                use="encoded"/>
 </wsdl:output>
</wsdl:operation>
<wsdl:operation name="validateNumber">
<wsdlsoap:operation soapAction="" />
        <wsdl:input name="validateNumberRequest">
                <wsdlsoap:body
                encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
                namespace="http://webservices.imacination.com/validate/Validate.jws"
                use="encoded"/>
        </wsdl:input>
        <wsdl:output name="validateNumberResponse">
                <wsdlsoap:body
                encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
                namespace="http://webservices.imacination.com/validate/Validate.jws"
                use="encoded"/>
        </wsdl:output>
</wsdl:operation>
</wsdl:binding>

<wsdl:service name="ValidateService">
        <wsdl:port binding="impl:ValidateSoapBinding" name="Validate">
            <wsdlsoap:address
            location="http://webservices.imacination.com/validate/Validate.jws"/>
        </wsdl:port>
</wsdl:service>
```

3.2.4. The Universal Description Discovery Integration (UDDI)

The Universal Description Discovery Integration (UDDI) framework is a registry service for Web services with service categorization features. It offers search engine capabilities for service locating specific services. UDDI itself is a Web service and message communication is based on SOAP; however it has no such limitation that only Web services are published in the registry or only services with a SOAP interface and a description based on WSDL. Merely, UDDI is a general-purpose registry. Although the conceptual UDDI structure conceptually resembles the Internet domain name server (DNS) service that translates Internet host names into TCP addresses, it is actually more like a replicated database service accessible over the Internet.

IBM, Microsoft and Ariba were the leading companies involved in the development of the UDDI specification as an Internet standard for Web service description registration and discovery. IBM, Microsoft, Hewlett-Packard, and SAP host the initial deployment of public UDDI services. Currently the UDDI specifications are developed by the UDDI Project, which is an independent consortium of companies, a partnership and cooperation that aims to define a standard for B2B interoperability (Gisolfi, 2001a). More than 300 other companies participate in the UDDI Advisors Group.

UDDI basic features are outlined in the UDDI Executive White Paper (Uddi.org, 2001) with the following statements:

"In summary, UDDI-based registries will help businesses to take advantage of Web services:

- Businesses will have a means to describe their services and business processes in a global, open environment on the Internet thus extending their reach.
- Potential trading partners will quickly and dynamically discover and interact with each other on the Internet via their preferred applications thus reducing time to market.

- The barriers to rapid participation in the global Internet economy will be lowered for any business anywhere
- Businesses will be able to organize their portfolio of services in a controlled environment."

UDDI provides three basic functions that are known as *publish, find, and bind*:

- **Publish.** How the provider of a Web service registers itself.
- **Find.** How an application finds a particular Web service.
- **Bind.** How an application connects to, and interacts with, a Web service after it has been found.

The UDDI specification includes a SOAP based Programming API that provides implementation features for a service broker. A service provider publishes a service in a service registry by making its WSDL description document available to potential requesters through Publisher APIs. Offerings are accessible to requesters at a basic programming-interface level. Service registry entries include a URL for the access of a specific service's WSDL file or other XML schema file describing the Web service. Inquiry APIs are used to discover registered services. A query can return multiple results. In case that one of the matching results is selected, a final API call is performed to get the needed contact information and the client can generate an appropriate message to send to the specified operation over the identified protocol.

A UDDI registry contains three kinds of information that is divided into groups in terms of telephone directories:

- **White pages.** Information such as the name, address, telephone number, short description, and other contact information of a given business.
- **Yellow pages.** Information that categorizes businesses. This is based on existing (non-electronic) standards.
- **Green pages.** Technical information about the Web services provided by a given business.

The key business registry data structures described in the UDDI specification are *business entities, business services, binding templates* and *service types.*

The UDDI business registry is created as a group of multiple operator sites. Operator sites are managed separately, however information contained within each registry is synchronized across all nodes.

Business entities describe information about a business, including their name, description, services offered, and contact information. Each business can also be associated with unique business identifiers, including a Thomas Register identifier and a D-U-N-S number, and with a list of categorizations that describe the business. New versions of the UDDI specification support additional categorization information.

A ***business service*** provides a description of the service offered by a business entity and information about categorization listings and bindings templates for that service.

Binding templates provide location information about the service and information on how to use the service. Each business service entry can have multiple binding templates, each describing a technical entry point for a service; for example, mailto, HTTP, FTP, fax, and phone. The binding template also associates the business service with a service type.

Structures called ***tModels*** describe what particular specifications or standards a service uses; in other words, they define service types. A ***service type*** defines an abstract service. Multiple businesses can offer the same type of service, all supporting the same service interface. A tModel specifies information such as the tModel name, the name of the organization that published the tModel, a list of categories that describe the tModel, and pointers to technical specifications for the tModel. With this information, a business can locate other services that are compatible with its own system.

Based on the data structure information, it is useful to go over the inquiry process again. The registry entry of a specific entry is a tModel entry for a certain service type. The

service binding and access point within a discovered WSDL document is the binding template entry. The relation between the outlined structures is shown in Figure 3.7.

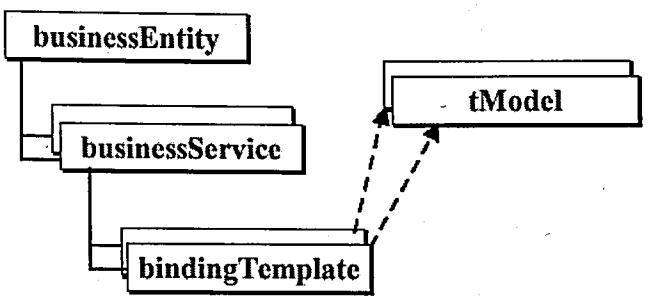

Figure 3.7. UDDI data model

There are public and private UDDI registries. The UDDI Project provides a global public registry, called the UDDI Business Registry. Individual enterprises can set up private registries to control and limit user access for service registry actions for enhanced security features such as data integrity and authorization.

UDDI specifies some business context descriptions including categorization information on the type of business, geographic location, and contact information. In addition, most registries have business classifications based on certain categories defined by organizations such as the North American Industry Classification System. Conclusively, the service search options include category-based search.

In many cases, human interaction through a web site is required to store input contact and service information, WSDL document locations etc. about a Web service on a UDDI directory service.

3.3. Current Shortcomings of the Web Services Architecture

3.3.1. Overview of Shortcomings

Core Web services standards, such as SOAP, WSDL and UDDI; already fulfill the promise of bridging disparate systems and platform domains. However, in their actual

state, Web Services are almost only used as API-centric applications with static server-side WSDL interfaces, functionally like pure client-server communications with RPC. But to reach up to the complete vision of Web services as enabling the use of application building blocks over the Internet, existing technologies need further extensions and enhancements (Newcomer, 2002). Except for the agreement on fundamental Web service standards, there is strong competition among Web service specification proposals and many commercial vendors come up with different opinions, which currently slow down standardization efforts.

Open Internet standards have several drawbacks for mission-critical applications as mentioned in the previous chapters. Specific features such as security, reliability, performance, QoS, support for business processes and transactions require more than core Web service standards, which are mainly concerned with data structures and static aspects at the communication level. Currently many companies prefer to deploy internal Web services and hesitate in adopting them for cross-organizational applications outside corporate firewalls before standardized security roadmaps have been set. Besides, extensive use of XML-based protocols can have a significant performance penalty. In addition, business level specifications for process and transaction support are still under development. For Web services facilitating seamless interoperability certain shortcomings must be addressed. There are several issues, which are agreed upon, to be in need of immediate improvements for enterprise-robust Web services (Bloor Research, 2002): Security/Privacy, Messaging/Routing, Quality-of-Service (QoS)/Reliability, Business Processes and Transactions, Management, Performance, Interoperability. These shortcomings are chief barriers to widespread external Web service usage. This section provides a detailed overview of these in-development areas, since they trigger many creative ideas and developments for further research and study.

3.3.2. Security

The lack of security mechanisms is a crucial factor limiting the external mainstream use of Web Services. Building blocks of security such as authentication, authorization, encryption, access control, and data integrity need standardized solutions instead of basic

standard recommendations. Furthermore, Web service security solutions must also cover policy, risk and trust issues. Currently, solutions for the identification, access restriction or authorization of a service provider or requestor are still in analysis or development phases.

In basic Web services transactions SOAP based data exchange for service requests is done in plain XML and clearly services are prone to data interception. As SOAP message transfer can use different transport protocols such as HTTP, SMTP, etc., the underlying security is not common for all types. A proposed security framework must apply to all transport protocols and security requirements must be abstracted from specific mechanisms already employed. Besides, such a framework must provide control options to handle legitimate intermediaries. It goes without saying that platform vendors, application developers, network and infrastructure providers, and customers have to work in coordination to establish a unified security approach despite the challenge of differing needs in heterogeneous environments.

The W3C Web Services Architecture Requirements lists a*uthentication, authorization, confidentiality,integrity, non-repudiation and accessibility* as the main security considerations for a comprehensive security framework (W3C, 2002c).

The Organization for the Advancement of Structured Information Standards (OASIS) works on several security related specifications. One of them is the Security Assertion Markup Language (SAML), a specification that deals with secure authentication, authorization and profile data exchange among partners. Another OASIS specification is the eXtensible Access Control Markup Language (XACML), which enables service access control for authenticated, authorized users. A low-level system standard is XML Key Management (XKMS), which offers features for the management of digital key and trust certification tokens available through which the complexity of Public Key Infrastructures (PKI) can be reduced. The W3C's XML Signature working group works on a system for digital signatures of XML documents. The XML Encryption working group works on encryption, including the XML Encryption Syntax and Processing specification, for encrypting data so that the result is a well-formed XML document (Ogbuji, 2002).

IBM and Microsoft have published another Web service security proposal, WS-Security, which has been announced as a "security roadmap for developing a set of Web Service Security specifications that address how to provide protection for messages exchanged in a Web service environment" (IBM, 2002a). The security plan describes how to utilize incompatible security technologies such as public key infrastructure, Kerberos, and others to resolve security aspects including authentication, authorization, privacy, trust, integrity, confidentiality, secure communications channels, federation, delegation and auditing.

3.3.3. Messaging/Routing

Data sharing and communication of enterprise-wide Web service applications rely heavily on messaging architectures. When extensive bi-directional message traffic takes place, efficient message routing and tracing operations are essential, especially in case of failure situations. Messaging operations span various communication patterns such as asynchronous one-way, request/response, broadcast, and conversational, or peer-to-peer. Additional Web services technologies also may depend on the messaging layer for certain qualities of service, such as reliable or guaranteed delivery, propagation of security and transaction contexts, and correctly routing messages along a defined path that includes one or more intermediaries.

There are already several messaging protocol initiatives. IBM has proposed reliable HTTP (HTTPR) to address requirements in this area. Another initiative WS-Inspection, which is an outcome of IBM and Microsoft collaboration, can be used to discover information about Web services available at a particular message target. Microsoft has also proposed WS-Referral and WS-Routing for definitions of a specific message path for a Web service, including any number of intermediaries, and how to route messages forward and backward along the specified route (Newcomer, 2002).

IETF defined the connection-oriented Internet protocol Blocks Extensible Exchange Protocol (BEEP). In addition, a SOAP mapping for BEEP has also been defined to extent SOAP messages with additional QoS features from BEEP, so that requestor/sender session

context can be maintained. Session contexts are useful for dealing with the transfer of multiple messages, which are from the same source or headed to the same target, as a single unit. Association with a connection is also possible through security and transaction contexts.

3.3.4. Quality-of-Service (QoS)/Reliability

The Internet has a dynamic infrastructure made up of a wide range of applications and services with differing characteristics and requirements, which compete for scarce network resources. Therefore delivering QoS on the Internet becomes a critical challenge (Anbazhagan and Nagarajan, 2002). Carrying out QoS standards can resolve network resource problems that occur due to heavy traffic, denial-of-service attacks, infrastructure failures, performance degradation of Web protocols, and security issues over the Web.

QoS features for Web services are non-functional service properties such as *performance, reliability, availability, and security*; features that can differentiate a service when compared with similar service offerings. These features are directly connected to security issues as the following section outlines. Initial Web service implementations only use WSDL to create a syntactic signature for the service interface, however QoS properties are not contained within the interface definition. A possible solution would be to describe these non-functional aspects with an additional Web service QoS specification.

Availability of a Web service represents a probability value that reflects if a service is ready for immediate use. The larger the value gets, the higher is the probability that a service is always ready for use. The time to repair (TTR) shows how fast a service is ready for reuse in case of failure and is directly related to availability. ***Accessibility*** and ***scalability*** are further QoS requirements for Web services. They determine to which extent service requests are successfully served. The ***integrity*** quality depends on correct execution and interaction of Web services with respect to the source. Integrity becomes especially important in transaction processing to specify successful completion or rollback of a process. The ***performance*** QoS property is measured in terms of throughput and latency. The result of serving a maximum number of requests is higher throughput.

Latency is the round-trip time between sending a request and receiving the response. ***Reliability*** of a Web service is determined by the capability of service and service quality maintenance. The number of failures on a periodical basis can reflect if message delivery during service requests and responses are executed as foreseen to measure reliability. The ***regulatory*** quality aspect is related with service conformance with rules, the law, standards compliance, and service level agreements (SLA). Faultless service invocation is only achieved when Web services strictly adhere to universal standards. Security requirements need to be addressed in case of QoS enabled Web services, as the public Internet provides no safe security approach.

3.3.5. Business Processes and Transactions

Within an enterprise not all operations are only based on information retrieval; some operations involve complex business processes and long-running transactions, which might interact within a certain process flow. The automation of process flow can enable transparent business interaction; however such automation requires orchestration features such as defining process relationships, execution order, sequences and tracing transaction messages. Therefore it would be favorable, that these features are also supported or contained within a business process centric Web service interface. Integration on a business process level has to provide dynamic process modeling and management options (Schmale, 2002).

Providing support for transactions enforces data consistency across business platforms that run a distributed process. For traditional integration techniques, this meant to construct a tightly coupled computing architecture to detect and overcome database locks, system or communication failures that occurred during transaction processing; a request that is not applicable on the Web. In addition, business processes and transactions are intertwined with Web service messaging architectures, due to rollback operations and complex messaging and message tracing requirements.

The first phase of Web services requires extra effort when business scenarios are implemented as applications wrapped with Web service features. A basic Web service

solution is not built on a specific transaction model for real-time transactions. An ideal framework would provide synchronous/asynchronous communication and interoperability among composite services, long-running transactions, joint exception handling and flow control in heterogeneous distributed computing environments.

Aside from the recent shortcomings in Web service like business processes, distributed business processing is yet an intricate issue for itself. Proprietary integration solutions do not provide compatibility between applications and platforms. Running business tasks on divergent platforms by multiple partners mostly ends up in interoperability problems, besides typical drawbacks like extended duration and communication latency. In the end, the integration solution adds new preconditions such as full task and activity monitoring, to make the system work. Attempting to implement real-time distributed, long-running business transactions behavior via Web services also challenges existing mechanisms. However, the dynamic infrastructure of Web services can help to establish a common framework for business process integration.

Despite already adopted Web service standards, the need for specifications that allow defining the business semantics of Web services arises. If business semantics are integrated within Web service implementations, services can be connected and Web service collections can be used jointly to define more complex functionality, such as specifying a business processes. A business process specifies the potential execution order of operations from a collection of Web services, the data shared between these Web services, which partners are involved and how they are involved in the business process, joint exception handling for collections of Web services, and other issues involving how multiple services and organizations participate (Leymann, et. al, 2002). A standardized business process framework for Web services would change the isolated and opaque nature of Web services.

Work on various business process languages and Web service orchestration proposals continue. These proposals try to form a specification that acts as glue for Web services that interact to accomplish a business process. OASIS designed the Business Transaction Protocol (BTP) proposal as a loosely coupled Web service interaction protocol. Web Service Flow Language (WSFL) by IBM and XML Language (XLANG) by

Microsoft merged into Business Process Execution Language for Web Services (BPEL4WS) supported by Microsoft, IBM and BEA. BPEL4WS has two complementary specifications, WS-Coordination (WS-COO) for message coordination and WS-Transaction (WS-TXN) for transaction handling. BPMI.org designed the Business Process Modeling Language (BPML) and Web Services Choreography Interface (WSCI) is backed by SUN (Assaf *et al.*, 2002).

3.3.6. Management

Web Service Management is merely the management of Web service compositions and their interactions. The management methodology applied is based on issues related to System and Application/Enterprise management categories within Network Management. In case of Network Management, system monitoring deals with fault, configuration, performance and security features.

Since the basis of distributed computing infrastructure is communication among disparate environments, monitoring communicating systems is an inevitable necessity. Without having insight into the actual system-state in a distributed environment, system administration and network monitoring becomes a daunting task. As for loosely coupled Web service technologies, there is a long way to go until sophisticated monitoring tools and utilities appear compared to tight-coupled computing system management utilities that exist today.

3.3.7. Performance

W3C principle focus is targeted at protocols and infrastructure; it does not work on specifications for tuning tools and utilities. This applies to active work conducted in Web service transaction processing, network/systems/application management and also performance areas. Only the W3C Quality Assurance works on developments for utilities and test suites in order to examine the load and scalability implications of standards as Web services and XML standards work together.

3.3.8. Interoperability

Due to the Web service hype, new vendors with Web service products or support enter the market steadily. The more diverse vendor platforms and implementations spread out, the more complicated and time-consuming it becomes to live up to Web service promised interoperability. Therefore achieving interoperability becomes a long-term goal (Bloor Research, 2002), as standard profiles and criteria for testing must be defined, obviously, based on actual conditions.

To meet universal Web Service Interoperability requirements, various Web service vendors and leading communities have established the Web Services Interoperability Organization (WS-I). The main concern is to respond to customer needs by providing guidance, recommended practices, and supporting resources for developing interoperable Web services so as to decrease the risk of Web services adoption in the long run. The organization releases public WS-I profiles, which are made up of use cases and usage scenarios, profiles, applications and testing tools. WS-I profiles are aimed to act as base prototypes that clarify the definition and usage of Web Services. Yet complex issues such as security, choreography or reliable messaging are beyond the scope of the published WS-I profiles. Advanced profiles can also be created as compositions of base profiles. However, in any case, security or choreography standards need to be finalized and stabilized, before specific advanced profiles are designed.

3.3.9. Final Thoughts on Web Service Shortcomings

The current limitations in the Web service landscape are serious obstacles for Web service adoption and external Web service deployment. It is expected that companies will follow the path taken by classic Web technology, along a continuum from low-value, non-mission-critical to high-value, mission-critical (Narsu and Murphy, 2002) to avoid risks. A safe approach would be to start Web service adoption with internal application integration, until standardization efforts for complex issues such as security, QoS and reliable messaging have been stabilized. Real business value can only be achieved in the long term,

when Web service technologies mature. Definitely, standardized Web service solutions and implementations have a direct impact on the duration of this time span.

4. WEB SERVICE ORCHESTRATION

4.1. Business Processes, Transactions and Workflows in Web Services

Recent advances in technology and business have made it unavoidable that enterprises coordinate the efforts of individuals and teams using different computer platforms residing on different locations. As in the case of E-Commerce, these locations are not restricted by enterprise boundaries. The coordination of knowledge, expertise share and business processes that span several organizations, often require the exchange of specific data objects among distributed applications. Users on heterogeneous platforms use the same data through standardized application interfaces. In case of managing business processes in large-scale systems, enterprises turn to workflow technology and business process management solutions as well. Due to the delegation of business processes across distinct systems, the provision of interoperability, application portability and transparency becomes a primary goal in system integration.

Realizing the interaction and interoperability of multiple business processes on heterogeneous systems can be accomplished by externalizing them as services and composing them into a new Web service. Although the Web service paradigm promises to connect divergent systems and convert traditional applications into "plug and play" services, critical enterprise applications often require complex tailor-made software solutions for exchanging data and processing intricate business scenarios. The Web service architecture lays stress on a service-centric view, but basically it originates from a technical perspective. In return, business processes have a completely business-oriented perspective. Therefore basic Web service standards are not sufficient to handle complex business processes and transactions. If companies want to embrace Web service technology as an alternative methodology to fulfill business integration requirements, definitely a new solution for automating business processes as Web services is needed. "Web Service Orchestration" provides the ability to prescribe how Web services are used to implement activities within a business process, how business processes are represented

as Web services, and also which business partners perform what parts of the actual business process (Leymann and Roller, 2002).

4.1.1. General Terminology

Figure 4.1 displays the key terms and interrelationships regarding business processes and workflow (WfMC, 1999). Terms addressed by Web Service Orchestration specifications are summarized in the next paragraphs. The main reference is a glossary published by the Workflow Management Coalition (WfMC), which is a nonprofit international organization founded by several vendors developing workflow management systems.

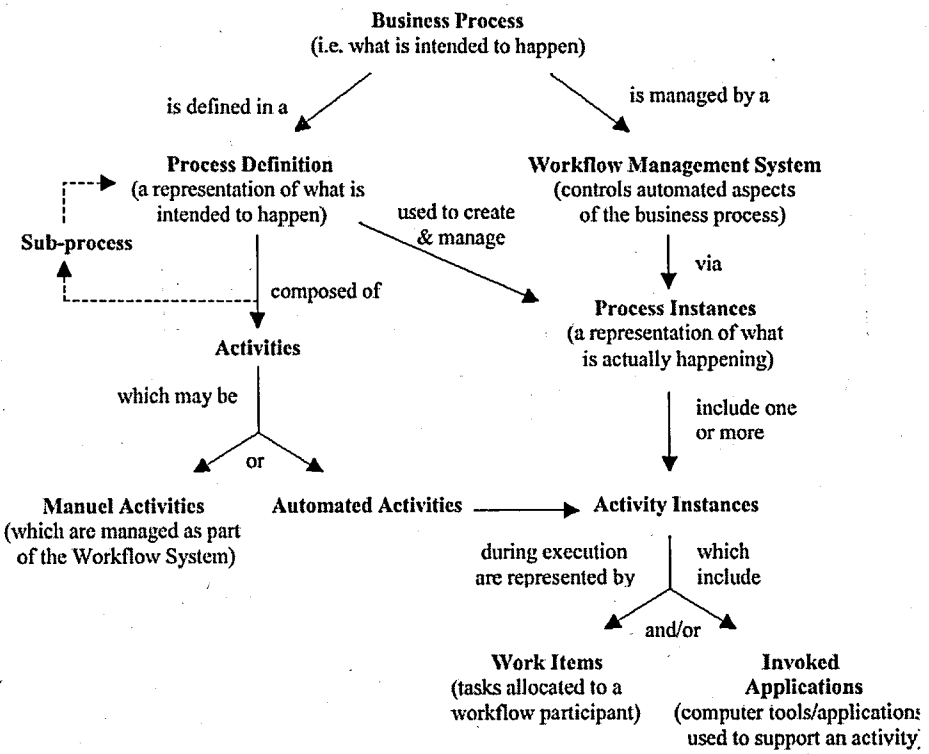

Figure 4.1. Business processing terms and their relations

Business processes are graphs of activities that carry out some meaningful business operation (Kreger, 2001). They are made up of a set of one or more linked procedures or activities which collectively realize a business objective or policy goal, normally within the

context of an organizational structure defining functional roles and relationships (WfMC, 1999). Processes can have starting and ending points and can also have a repetitive structure. Credit checking, purchasing a concert ticket, managing inventory in a warehouse can be given as examples. Business processes vary in level of complexity and functional boundaries. Some processes may consist of a single step or an atomic transaction, whereas long-running transactions that require tracking and rollback features can also constitute a process.

A ***process definition*** is the representation of a business process in a form, which supports automated manipulation, such as modelling, or enactment by a workflow management system. The process definition consists of a network of activities and their relationships, criteria to indicate the start and termination of the process, and information about the individual activities, such as participants, associated IT applications and data, etc (WfMC, 1999). A ***process instance*** is the representation of a single enactment of a process including its associated data (WfMC, 1999), which is basically the process definition.

An ***activity*** is a description of a piece of work that forms one logical step within a process. An activity may be a manual activity, which does not support computer automation, or a workflow (automated) activity. A workflow activity requires human and/or machine resource(s) to support process execution: where human resource is required, an activity is allocated to a workflow participant (WfMC, 1999). An ***activity instance*** is the representation of an activity within a single enactment of a process, i.e., within a process instance including its associated data (WfMC, 1999).

Workflows are business processes that are based on document lifecycles and forms-based information processing and execute in an IT environment according to a set of procedural rules. Workflow tools allow businesses to define each of their business processes as a series of activities carried out by individuals or applications, and vary the sequence through the activity series depending on the output data from each individual activity. A significant limitation of workflow implementations is that they are closely tied to the businesses in which they are deployed, and cannot be reliably extended to customers, suppliers and other partners.

Composing Web services can become a variant of a workflow setup, since services can dynamically be orchestrated into flows and make multiple activities interact with each other across a network of distributed processes. The goal of Web services workflow is to enable the same type of seamless integration across business processes and transaction lifecycles that make use of many Web services (Snell, 2001).

4.2. Web Service Orchestration

A business integration infrastructure for Web services turns into a key requirement in areas such as administration, insurance, banking, including industrial and manufacturing applications, etc. Such Web services have to provide synchronous/asynchronous communication and interoperability among composite services, long-running transactions, joint exception handling and flow control in heterogeneous distributed computing environments. The complete set of these actions is covered by the term "Web Service Orchestration". In brief, Web services orchestration is about providing an open, standards-based approach for connecting web services together to create higher-level business processes (Peltz, 2003).

Web Service Orchestration specifications are developed to form a basis for linking or composing services together into business processes. The essential idea behind these specifications is to extend the basic Web Services stack with a so-called business process integration layer and make Web services support business operations. A general list of a variety of Web Service Specification proposals can be seen in Table 4.1.

Table 4.1. Emerging business process description languages

Emerging Business Process Description Languages		
Specification	**Companies**	**Version/Date of Release**
BPEL4WS Business Process Execution Language for Web Services	BEA, IBM, Microsoft	V.1 - August 2002
	BEA, IBM, Microsoft, SAP	V1.1 - April 2003
BPML Business Process Management Language	BPMI.org	V.1 - June 2002
ebXML Electronic Business XML	OASIS, UN/CEFACT	V.1.04 -February 2001
WSCI Web Service Choreography Interface	BEA, Intalio, SAP, SUN	V.1 - June 2002
WSFL Web Services Flow Language	IBM	V.1 - May 2001
XLANG	Microsoft	V.1 – 2001

4.2.1. Orchestration vs. Choreography

The terms orchestration and choreography are generally used interchangeably. In fact, the former describes how to coordinate and execute message flow and Web service interaction, especially in case of multiple services making up a business process or long-running transactions in an actual service implementation. For that reason, orchestration does not handle output variations or errors that occur during actual execution. The latter is mainly involved in tracking public message flow of Web service interaction in a top-down view. Choreography defines behaviors for handling varied and unpredictable interactions among a set of Web services (Virdell, 2003) and represents a higher-level syntax. Though it seems that within time, both approaches will merge into a single meaning.

The definition of the term choreography is still in progress in the Web Services Glossary of the World Wide Web Consortium (W3C). The current description is outlined as (W3C, 2003): "*The specification of the ordering of messages from one node's perspective or a collection of nodes. May or may not include Turing complete logic in determination of the message exchange pattern.*"

4.2.2. Web Service Orchestration Specifications

The following sections outline existing business process language specifications. Comparative results are mainly presented for BPEL4WS, WSCI and BPML. As WSFL and XLANG have been superseded by the BPEL4WS specification, only a brief overview is made. The other specifications have been included in the review so as to provide a general insight of the variety of solution proposals, however they are not the central focus point of the comparison study.

4.3. Web Services Flow Language (WSFL)

IBM published the Web Services Flow Language (WSFL) proposal in 2001; the specification can be found at (Leymann, 2001). WSFL is an XML language for the description of Web Services compositions, which can be either public or private process flows. WSFL is not concerned with modeling business processes.

The purpose of a WSFL document is to define the composition of Web Services as a flow model or a global model. The *WSFL Flow Model* is the actual XML representation of a directed graph approach for defining and executing business processes. The model is used to compose Web services, as defined by their individual Web Services Description Language (WSDL) documents, into workflows. Flow models are also named as *flow composition, orchestration or choreography*. The *global model* binds activities with public interfaces that allow business processes to advertise themselves as Web services. The <globalModel> element defines the identity, the location and the implementation of the service provider that implements a specific role.

Both the flow and global model have a declared public interface and an internal compositional structure. The composition assumes that the Web services being composed support certain public interfaces, which can be specified as a single port type or as a collection of port types. These collections are called service provider type, which must properly implement the appropriate Web service interface in order to be classified as the appropriate type of service provider to handle a particular activity in the business process.

A WSFL document uses serviceProviderType' elements to identify the roles of the implementers of specific activities within the context of a given business process model. The serviceProviderType is defined by a WSDL document.

A business process consists of four kinds of WSFL elements: activities, control links, data links, and plug links. The <flowmodel> element is an abstract definition of the workflow process. It contains <activity> elements, which define activities that are implemented in the form of a Web service defined by WSDL. The flow model also contains control links and data links in a decoupled fashion. Control links define the execution sequence for activities or may also specify transition conditions. Though they do not provide structural information for data exchanges, since data links are responsible how message exchanges are performed. Plug links are used in WSFL to model the interaction between remote service providers.

General notes about WSFL:

- WSFL is integrated with UDDI and WSDL for dynamic selection of Web services.
- Once the global and the flow model for a given business process are defined, the whole business process can be defined as a single Web service that may be used by other business processes. In brief, a WSFL definition can also be exposed with a WSDL interface, allowing for recursive decomposition.
- WSFL supports exception handling.
- WSFL has no direct support for transactions.

4.4. XLANG

In 2001, Microsoft released the first draft of the XLANG language, which was initially developed for the Microsoft BizTalk Server. XLANG aims to serve as the basis for automated protocol engines that can track the state of process instances and help enforce protocol correctness in message flows (Thatte, 2001). In brief, the goal of XLANG is to make it possible to formally specify business processes as stateful long-running interactions between Web service providers.

An XLANG service is described by WSDL. Instances of XLANG services are started by operational implicit values or specific activation operations. The interactions between services occur through message exchanges expressed as WSDL operations. The behavior of an XLANG service is composed of actions. These actions can be WSDL operations, delays, and exceptions. With various language constructs such as <empty>, <while>, <sequence>, <switch>, <pick>, <all>, <context> and <compensate> action compositions can be produced. In XLANG the focus is on publicly visible in processes in the form of messages exchanged. General XLANG language constructs are listed below:

- **Basic Control Processes.** These processes include sequential and parallel control flow constructs. A <sequence> element contains serial actions or processes to be performed. The <switch> process is used to specify conditional operations. The <while> construct introduces loops. The <all> process executes all sub-processes concurrently. The <pick> process awaits a specific message (action) to execute. The <empty> process acts as a null element.
- **Contexts.** The <context> construct provides a framework for local declarations, exception handling and transactional behavior. Currently only local declaration of correlation sets and port references are allowed. XLANG provides a robust exception handling mechanism. Compensating a process can be associated with a context in case of transactional operations.
- **Long running transactions with compensation.** XLANG provides transactional features through a long-running transaction model, however does not support coordinated transactions. The <compensate> element can be used for compensation of transactions. This element invokes the compensation process block of an enclosed transaction. Compensation is performed for processes and may need to be run in a specified order.
- **Correlation and port references.** XLANG has limited data manipulation capabilities except for correlation sets and dynamic port references. As interaction is an instantiation of a service, exchanged messages not only need to be delivered to the correct destination port, but also to the correct instance of the service that defines the port. Therefore, each message contains an instance dependent 'correlation token'. It is possible to specify correlated groups of operations within a service instance. A set

of correlation tokens can be defined as a set of properties shared by all messages in the correlated group. Such a set of properties is called a correlation set. The set of operations (in a single service) that is correlated via a correlation set is called a correlation group.

- **Dynamic service referral.** XLANG provides for passing and dynamic binding of port references for dynamic target referral.
- **Business Process Contracts.** An XLANG service description can define the behavior of business process participants, which, in other terms, take part in a contract. The description is not business-oriented; it is basically a mapping of interacting ports. Besides, the port types are constrained to contain only incoming, or only outgoing operations. In the end, these features do not support business transaction, non-repudiation, or legally binding transactions.

4.5. Business Process Execution Language For Web Services (BPEL4WS)

4.5.1. Introduction to BPEL4WS

Business Process Execution Language for Web Services (BPEL4WS) is an XML-based business process composition language for aggregating web services into abstract or executable business operations, which both consume and provide Web services. The language can be used to define Web service interaction and cooperation protocols.

BPEL4WS provides support for executable complex business processes, simple abstract processes and it can also deal with long-running transactions. It is possible to create stateful business process instances by exploiting correlation mechanisms of BPEL4WS. Service partners, service link types and service references can be determined. Service invocation, fault handling, process termination, structured activities based on sequences or conditions are some of the other concepts covered by the BPEL4WS specification. Furthermore, BPEL4WS offers support for process communication, automation, coordination and partner integration for Web services.

BPEL4WS business processes can be executed and ported across BPEL4WS-conformant environments with the use of a BPEL4WS engine. They can also interoperate with partner Web services, which are not implemented based on BPEL4WS (Leymann *et al.*, 2002). BPEL4WS language can be extended with additional constructs from other XML namespaces (Curbera *et al.*, 2002). Through the BPEL4WS framework for business-process applications in distributed systems, real life business cases can also benefit from the advantages of Web Services.

4.5.2. BPEL4WS - Convergence of WSFL and XLANG

BEA, IBM and Microsoft released the first draft of the BPEL4WS specification in July 2002. BPEL4WS represents the cooperative merging of IBM's Web Services Flow Language and Microsoft's XLANG specifications superseding both these specifications. BPEL4WS combines graph-oriented process features of WSFL and XLANG support for structural processes constructs into a single solution for Web service orchestration, workflow and composition. The most recent development is that BEA, IBM, Microsoft, SAP have published the (BPEL4WS) V1.1 specification and intend to submit it to the OASIS Web Services Business Process Execution Language Technical Committee.

4.5.3. BPEL4WS and WSDL

BPEL4WS derives from the following XML-based specifications: WSDL 1.1, XML Schema 1.0, XPath 1.0. BPEL4WS is layered on top of WSDL and uses WSDL to specify actions that take place in a business process, and their sequences. The role of BPEL4WS is to define a new Web service by composing a set of existing services. Input and output points are described by WSDL. The interface of the composite service is described as a collection of WSDL portTypes, just like any other Web service. Interfaces contain WSDL data types and can reference external services through WSDL descriptions. An illustration of a BPEL4WS flow can be seen in Figure 4.2 (Weerawarana and Curbera, 2002).

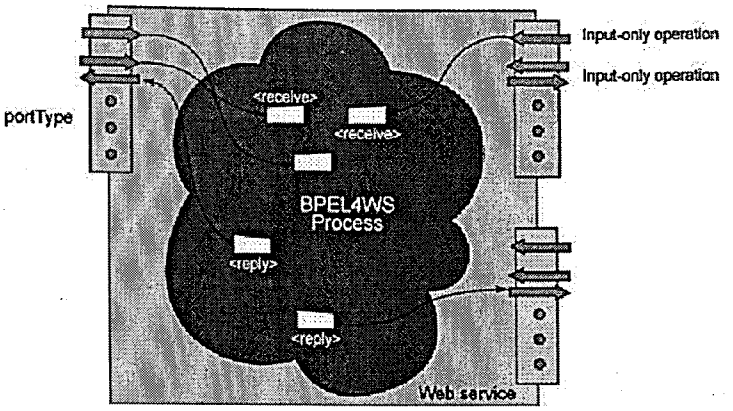

Figure 4.2. BPEL4WS flow

WSDL initially supports an interaction model with stateless client-server model of synchronous or uncorrelated asynchronous interactions, whereas BPEL4WS business processes represent stateful long-running interactions in which each interaction has a beginning, defined behavior during its lifetime, and an end (Curbera et. al, 2002).

4.5.4. BPLE4WS - Complementary Specifications

BPEL4WS has two complementary specifications, WS-Coordination and WS-Transaction, also developed jointly by IBM and Microsoft. They complement BPEL4WS in that they provide a Web services-based approach to improving the dependability of automated, long running business transactions in an extensible, interoperable way for a distributed computing model. The ***WS-Coordination*** specification defines a framework through which those services can work from a shared "coordination context". That context contains the information necessary to link the various activities. The ***WS-Transaction*** specification, on the other hand, provides a framework that allows monitoring the success or failure of each individual, coordinated activity. It specifies a protocol for the long running transaction model defined in BPEL4WS as well as atomic transactions between regular Web services. The fact that WS-Transaction is based on Web services means that transaction support can be made interoperable across vendor-specific transaction management applications. Since the specifications act as an explicit layer within the Web service stack, they provide extensibility features and can be customized.

The complete package of BPEL4WS, WS-Transaction, and WS-Coordination specifications claim to be a comprehensive business process automation framework that helps companies to create and automate business transactions in order to streamline business integration functions. WS-Coordination and WS-Transaction are building blocks for enabling consistency in distributed web service operations. Both of these specifications can be used with other application-specific protocols across vendor implementations, trust domains to provide web service interoperability.

An example interaction of the BPEL4WS, WS-Coordination and WS-Transaction specifications would be as follows (Snell, 2002):

- Definition of how to integrate services into business processes
- Definition of how specific activities of a business process can be externalized publicly as Web services
- Coordinating the activity of multiple Web services within the overall business transaction.
- Dynamically linking to services from multiple providers at runtime based on data derived from the process flow itself.

4.5.5. BPEL4WS Specification

The ***BPEL4WS process*** can be described as a flowchart like expression of an algorithm (Weerawarana and Curbera, 2002). The main elements of the BPEL4WS are displayed in Table 4.2 and explained in details in the following sections.

Table 4.2. BPEL4WS language concepts

General elements	Basic Activities	Structured Activities
Partners	Terminate	Flow
Containers	Receive-Reply-Invoke	While
Property	Throw-Compensate	Sequence
Fault handler	Assign	Scope
Compensation Handler	Wait	Pick
Source-Target	Empty	Switch
Service references		
Service link types		
Service links		

4.5.6. Basic Activities

Actions that constitute a business process are called ***activities***. Basic activities include non-sequenced operational steps such as passing data or handling exceptions. As they are non-sequenced, they cannot enclose other activities.

The <terminate> construct can be used to immediately terminate a business process.

Receive, reply and ***invoke*** activities are used for interaction with the outside world and each of these activities identifies the Web services call it belongs to by specifying a portType, operation, and partner. Receive and reply form a request-response operation.

The <receive> construct makes business process do blocking-wait for a matching message to arrive.

The <reply> construct allows the business process to send a message in reply to a message that was received through a <receive>. Within a process multiple <reply> can be defined, however only one of them will become active.

The <invoke> construct allows the business process to invoke a one-way or request-response operation on a portType offered by a partner. It specifies input and output

containers, for the input and output of the operation being invoked. An invocation can be either synchronous (request/response) or asynchronous (one-way). In the latter case, only an input container is required.

The <throw> construct can be used to generate a fault from inside a business process. Faults must have unique names within defined activities to be caught. Optional containers can be used to send the requestor specific fault information.

The <empty> construct is used to insert a "no-op" instruction into a process so that the activity takes no action. It may be used for suppressing a fault or for synchronization of parallel activities, for instance.

The <assign> construct can be used to make specific value assignments for updating or inserting data. It contains <copy> elements, which include <from> and <to> elements. Assignments have to be made with type-compatible data. There are several forms of the <assignment> construct.

The <wait> construct can be used to make a process wait for a specific period until a certain deadline, which has been specified as an expiration criteria, is reached.

The <compensate> construct is only invoked within a fault or compensation handler to undo the effects of already completed activities.

4.5.7. Structured Activities

Structured activities provide concurrency and synchronization so that basic actions can express complex processes. They support definitions for control patterns, data flow, fault handling and message coordination.

The <while> construct provides a looping mechanism to repeat a certain activity based on a specified criterion.

The <flow> construct provides a parallel execution mechanism so that multiple activities run parallel. By defining links, with nested source and target activities, explicit controls can be performed. Literally, a flow is a directed graph with the activities as nodes and so-called links as edges connecting the activities in BPEL4WS (Leymann and Roller, 2002). An activity specifies the links it is a source or target of.

The <scope> construct can be used to group activities and define common execution contexts for fault and compensation handling. Scopes can be used to create long-running transactions.

The <sequence> construct is a placeholder for sequential activities. If activities have been completed in their lexical order, a sequence completes.

The <pick> construct can be used to make a process wait until exactly a specific message arrives or wait for a time duration and then trigger the activity for execution. The event handlers within a <pick> construct include alarm (onAlarm) and message handlers (onMessage) for triggering activities. If an activity is executed, <pick> completes. In brief, <pick> is a structured version of <receive>. Besides, business process descriptions must start with a <pick> or <receive> construct. Both of these constructs have an optional boolean createInstance flag, that indicates if an instance of the specified business process should be created if none exists already. As a result, developers do not need to know if a process has been instantiated to interact with a BPEL4WS process.

The <switch> construct provides a case-selection logic so that only an activity that satisfies a given condition within a branch of activities is executed. The list of activities is wrapped with a <case> construct, which can be followed by an optional <otherwise> construct. The <case> branches specify Boolean XPath expressions to be evaluated in the given order. The first <case> element whose Boolean expression evaluates to true has its child activity performed.

4.5.8. BPEL4WS - General Elements

Partners are services that interact with each other during the execution of a business process. Each partner is characterized by a serviceLinkType, which specifies the roles of partners. The same serviceLinkType can be used for multiple partners.

Service references are used to represent dynamic communication data, which is also called port-specific data, to select a provider for a particular type of service and to invoke offered operations. Partners are assigned unique service references during process deployment or execution.

Service links are used to define partner-relationships through bi-directional message and port type definitions.

Data such as routing information can be stored within a business ***context***, which is defined as a container. Basically, ***containers*** are WSLD messages exchanges across partners. To avoid data loss, containers can also be stored persistently.

Properties are correlation data, which are defined as separate entities in BPEL4WS so that different messages can use the same property. A mechanism called *aliasing* can be used to point to a property field within a message.

Fault handlers detect errors within a certain scope and throw a fault message through WSDL ports or they start a recovery action. ***Compensation handlers*** are used to undo already completed activities. The innermost scope tries to correct a fault; if it does not succeed the fault will be thrown to an outer scope.

4.5.9. Abstract and Executable Processes in BPEL4WS

Abstract business processes can be used to define business protocols (views), which are not executable in general and which do not include internal structure details. Business protocols define the ordering in which a particular partner sends messages to and expects

messages from its partners based on actual business context without revealing internal behavior (Curbera *et al.*, 2002).

Executable business processes model actual behavior of a participant in a business interaction without separating internal and external details of a process. Process engines can interpret executable BPEL4WS scripts so that modeling business activities is sufficient to implement business processes.

4.5.10. Transactions in BPEL4WS

The <scope> construct in BPEL4WS can be used to define transactions. As mentioned in the previous section, it provides fault and compensation handling operations for nested activities so that activities can be collectively undone. Parent and child scopes use WS-Transaction in order to determine the outcome of a long-running transaction. Currently BPEL4WS supports only long-running transactions within a single process hosted on a single BPEL4WS engine; however future extensions may support transactions across distributed processes and BPEL4WS engines.

4.5.11. Changes in BPEL4WS 1.1

- **Core concept clarification.** BPEL4WS can be used to describe a public business protocol as well as an executable business process. The separation of core concepts from extensions in BPEL4WS 1.1 allows features required for specific usage patterns to be defined in a composable manner. This can help to clarify the difference between choreography (description of a public protocol) and orchestration (description of an executable process).
- **Scoped variable and correlation sets.** In BPEL4WS 1.1, the limitation of defining variables (called "containers") only globally at the process level has been lifted. This allows developers to define local variables and minimize the amount of data that needs to be carried throughout the lifecycle of a process so that enhanced performance can be achieved.

- **Event handlers.** Event Handlers allow developers to define logic about how unscheduled events and requests should be handled as part of the flow. Those unscheduled events are processed in parallel with the main activity. For example, this feature can be used to define how an order cancellation event can be processed as part of an order management flow. Events allow for the implementation of more dynamic business processes.

4.6. Web Service Choreography Interface (WSCI)

4.6.1. Introduction to WSCI

The Web Service Choreography Interface (WSCI) is an XML-based interface description language that describes the message-exchange flows for Web service interaction and the dynamic interface of each service by providing behavioral description of service interfaces. In brief, WSCI is mainly concerned with interdependencies among Web services. In June 2002, BEA, Intalio, SAP and Sun Microsystems have released the WSCI 1.0 draft specification.

WSCI identifies message correlation, transaction description, exception handling, thread management, properties and selectors, connectors, operational context, dynamic participation as the key requirements for Web services choreography. With WSCI it is possible to keep track of message types, sequences, relations and exceptions within a business process flow made up of multiple services interacting with each other. These services can be based on intra-organizational or external Web service implementations. WSCI defines standard programming language constructs such as sequences, loops, exception handling, and conditional controls.

When analyzing WSCI, the following points have to be emphasized:

- WSCI cannot be categorized as a workflow language; however it can describe Web service interaction within a system with workflow features.

- It does not define business processes for Web services or the internal behavior of such processes.
- It does not act as a middleware in a message-exchange.
- It does not directly determine message flow.
- The use of some internal software within a Web service is out of scope for WSCI.
- It is not an executable language.
- It is independent of integration models within existing services.
- Although it is directly related with message flow of Web services, internal operations of a service are irrelevant to WSCI.

4.6.2. WSCI within the existing Web Services Stack

The initial Web Services stack layer consists of Web services messaging and description standards, which are SOAP and WSDL respectively. The emerging higher-level standards make up a modeling layer that is in charge of business processes, transactions and collaborations. WSCI acts as a gateway, as glue connecting these separate Web service standard's layers. Its goal is to establish a common choreography framework linking these layers to enable interoperability for interaction. For this matter, it is appropriate to consider WSCI as a medium at the operational level acting with a bottom-up approach as it is illustrated in Figure 4.3.

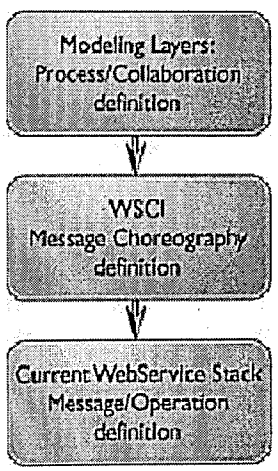

Figure 4.3. WSCI within the Web services stack

WSCI is designed to work on top of Web Service description languages, mainly WSDL, to turn a static Web service interface into a dynamic, choreographed interface. Aside from the description layer, WSCI acts below process or collaboration modeling layers within the emerging Web Service stack.

WSDL operation descriptions are opaque, whereas WSCI supports complex descriptions across multiple WSDL descriptions. Even though a description language provides only a single interface to a Web service, a Web service can exploit multiple WSCI interfaces for different participants of a message exchange activity.

4.6.3. WSCI Specification

The main source for WSCI documentation is its specification and there are not many articles or examples to analyze the language features. Therefore the following section provides a generic overview of WSCI language constructs referencing its released specification 1.0 (Assaf *et al.*, 2002). Table 4.3 lists the topics that have been covered.

Table 4.3. WSCI language concepts

Language Concepts
Global Model, Interface, Activities, Choreography, Processes, Properties, Context, Message Correlation, Conversation, Exceptions, Transactions

Before going into further detail, basic notes about the language are presented below:

- WSCI reuses the wsdl:definitions and the wsdl:import elements.
- The <documentation> element provides human-readable information.
- The selector definition is a referencee to a data type from some type system.
- Top-level definitions include interface, selector, correlation and model elements.

4.6.4. WSCI Global Model

The ***WSCI Global Model*** can be used to create a visual model of a service by describing message exchange flow, participants, operations and operational links of a process.

Descriptions are based on WSCI interface elements. Literally, the model is a logical abstraction of a service. It can prove to be a useful tool, especially for visual modeling, analysis, validation and simulation of the overall process.

The syntax for the WSCI Global Model includes <model>, <interface> and <connect> language elements. The <model> element must contain at least two references to interfaces of participants, and one or more links between operations of communicating participants. The <interface> element references a WSCI interface. The <connect> element describes a link between operations of communicating participants.

4.6.5. WSCI Language Constructs

A WSCI ***interface*** describes the message exchange behavior of a service. Exposing multiple interfaces is allowed. For that reason, a service can support multiple scenarios or perform different for a single scenario in specific cases. The <interface> construct is a top-level WSCI definition and appears therefore as child of the wsdl:definitions element.

Activities are operations performed by Web services. WSCI activities are atomic or complex. Atomic activities represent basic operations, whereas complex activities can be compositions of multiple activities.

In WSCI ***choreography*** describes temporal and/or logical dependencies among activities. Activity categories and related language constructs are listed below:

- **Sequential execution.** The activities must be executed in sequential order. The complex <sequence> activity is used to run sequential activities in a given order.

- **Parallel execution.** All activities must be executed, but they may be executed in any order. The complex <all> activity performs all the activities within an activity set if possible in parallel.
- **Looping.** The activities are repeatedly executed based on the evaluation of a condition or an expression. The complex <foreach> activity is for-loop construct to perform all activities repeatedly. The complex <while> activity is used to perform activities repeatedly based on a specific conditional value. The complex <until> activity performs all the activities repeatedly, at least one or more times, based on a conditional value.
- **Conditional execution.** One out of several sets of activities is executed based on the evaluation of conditions (switch) or based on the occurrence of an event (choice). The complex <switch> activity is selects one activity set from a collection of one or more activity sets based on a set of conditions. The complex <choice> activity selects one activity set from a collection of two or more activity sets based on the first event triggered.

There are also additional activities:

- The atomic <delay> activity is used to represent a time interval.
- The atomic <empty> activity is used to define a no-operation action.
- The atomic <fault> activity triggers a fault in the current context.

Multiple activities that execute within the same context make up an ***activity set***, which can include any type of activity element. It is also possible to use activities defined in other specification as long as they are defined in a namespace that is other than WSCI and use the wsci:activity abstract element as their substitution group.

Another concept in atomic activity definitions is the <actions> element, which describes how operations, especially message exchanges, are performed. Actions are based on WSDL operation types such as one-way, request-response, notification and solicit-response.

A ***process*** is a reusable WSCI unit of reuse. There are several methods for process instantiation. A process can be triggered by the messages it defines or from within a service implementation. The process can also be called, similar to function calls in other programming languages, by the atomic <call> activity or it can be spawned so that it is executed as a parallel thread by the atomic <spawn> activity. There is also an atomic <join> activity that waits for instances of a spawned process to complete. The <spawn> and <join> constructs can handle the use of multiple parallel threads.

WSCI process types are categorized as top-level and nested processes. ***Top-level processes*** are defined at the interface level and can be accessed globally within the interface. ***Nested processes*** are defined within complex activities and nested process can be referenced only from within the complex activity that defines it. For a top-level process definition, the <process> definition is placed within the <interface> element. To define a nested process, the <process> definition is placed within the <context> element.

It is important to distinguish the process concept in syntax, design-time and runtime definitions. The term "process construct" refers to the WSCI syntax. The term "process definition" refers to an occurrence of the process element within a WSCI document. The term "process instance" or “process” refers to the conceptual instance of such a definition.

Properties are basically values that can be of any data type. They are used to reference specific message or service values.

A ***context*** describes the environment in which a specific set of activities is executed. These descriptions include information available to the activities, exceptional events and recovery behavior and transactional properties. Definitions can be local properties or local process definitions. The former is only available to the activities executing in this context, whereas the latter designates exactly those processes that may be instantiated in this context.

4.6.6. Message Correlation

A service can take part in multiple message exchanges, so-called conversations, at the same time. ***Correlation*** is a control mechanism for structuring conversations in order to protect message consistency. A correlation instance is a set of properties' values and it can be used to identify different conversations.

The corresponding language constructs are <correlation> and <correlate>. The <correlation> element, which is a WSCI top-level definition, can be used to specify properties for defining the correlation identity. The correlate element is used to relate an action with the correlation definition.

4.6.7. Exception Handling

There are three types of exceptions in WSCI: Exception message, Fault occurrence and Timeout; which are defined by the <onMessage>, <OnFault> and <onTimeout> elements respectively. Exception handling can include complete or partial termination and recovery within the current context.

4.6.8. Transactions

WSCI transactions are either atomic or open-nested. ***Atomic transactions*** have no further inner structure. Transactional message exchanges must be coordinated based on a certain concept like (e.g. CORBA OTS, JTS). There is no specific rollback feature provided by WSCI. ***Open-nested transactions*** are transaction compositions of atomic or open-nested transactions. Rollback for such transactions is done in a recursive manner.

Since atomic transactions require resource locking, they are only recommended for short-lived transactions. On the contrary, open-nested transactions can be used for long-running transactions, as they require short resource locking for each single unit of action that is contained within the nested structure.

Rollback operations undo transactional changes when a transaction is aborted but not completed, while compensation operations undo changes after the completion of a transaction. The atomic <compensate> activity is used to recover completed transactional actions. It can only be used in the parent context of a completed transaction. In addition, a transaction instance can only be compensated once.

4.6.9. WSCI Extensibility

The WSCI extensibility model supports introducing additional semantics and expanding WSCI interface and model definitions through extensibility elements. Additional semantics can be new activity types, extended actions or WSCI constructs used within other specification documents.

4.6.10. WSCI – Future Work

The main concern of possible future enhancements is directly related with collaboration features of Web services.

- **Exclusion Groups.** The possibility of specifying properties indicating services that must be excluded from a set of services, possibly though a value that is a URI, can be included in the specification.
- **Web based Collaboration.** Issues like Contract, State, Role, Rights and Obligations of Participants will be addressed in future implementations to support the evolving Collaboration layer within the Web services stack. Facilitating these issues may require syntactic changes, however the WSCI specification underlines the ultimate goal of web services as "enabling web-based collaboration".
- **WSCI Global Model.** The current WSCI specification defines operations that are tightly-coupled to WSDL operations due to the fact that initial Web services are of atomic nature. Probably future versions of the specification will allow looser coupling, global mapping of WSCI actions and greater re-usability of operations across multiple actions and. As a result collaboration issues can be facilitated when the work in this area matures.

4.7. Business Process Modeling Language (BPML)

4.7.1. Introduction to BPML

The Business Process Modeling Language (BPML) specification provides an XML-based declarative scripting language meant to be used with a workflow engine. The ultimate goal of the language is to enable separation of business logic from execution logic. The specification supports modeling and execution of business processes within its abstract structure. BPML can deal with basic atomic transactions and complicated business processes. The basic concepts the model claims to provide, are end-to-end processing support, transparent process data persistence, data management, dynamic control flow, embedded business rules, nested processes, distributed transactions, exception handling and a mathematical model that is based on the Pi-Calculus Mathematical Model. BPML makes use of the Web Service Choreography Interface for public interface descriptions and choreography, as both specifications have a similar execution model and syntax.

BPML was created by the Business Process Management Initiative (BPMI) and emerged from research conducted by various companies. A final draft has been released in November 2002. BPMI.org works on other complementary specifications. ***Business Process Query Language (BPQL)*** is used for the deployment, control and optimization of business processes. ***Business Process Modeling Notation (BPMN)*** is used for the design of business processes.

The following points outline important language concepts:

- Language constructs include support for basic, process flow and structured activities. Partner and role definitions are also allowed. Structured activities can handle conditional choices, sequential and parallel activities, joins, and looping. Task scheduling is possible.
- Timeout constraints can also be specified for specific activities defined within the process.

- Persistence, roles, instance correlation, and recursive decomposition features are available. Recursive composition is the ability to compose sub-processes into a larger business process through nested processes.
- Transactional support and robust exception handling mechanisms are provided. The language can deal with short and long-running transactions. The compensation logic for transactions uses scope definitions. Activity timeout constraints can also be used to quit a process.
- Nested processes and transactions can be described by BPML.

4.7.2. BPML Specification

This section provides a detailed overview of BPML concepts; it references information from (O'Riordan, 2002). The original BPML specification can be found at (Arkin, 2002).

The BPML language concepts covers *collaboration, workflow, transaction management, exception handling and service interface issues.*

BPML is **collaboration-based** and employs a message-based model to describe processes, participant interaction and message flow. XML schemas are used to define the structure and type of the message content. Participants can be abstracted using roles and represent various entities such as organizations, applications, users, other processes, etc. Participants can be assigned statically or determined dynamically at runtime.

In essence the private process represented by a BPML process interacts with participants through a set of collaborations. Such descriptions are amenable to generation from collaboration modeling tools.

BPML provides comprehensive control and data flow support for **workflow**. A process can consists of a simple, complex or process activity. Simple activities are used to model message exchanges, tasks, data, or failure. Complex activities include block-structured control flow constructs for serial, parallel, and conditional execution of multiple

activities. They can also model compound states, consisting of multiple sub-activities representing sub-states. Process activities manage data to spawn and join nested processes, to suspend and complete the process, and to repeat activities or states, which allow to set up conditional and non-conditional loops. Join patterns enable synchronous processes and activity coordination across multiple participants. BPML supports a process selection and participant branching for such processes.

Activities can be scheduled to start at a future date, and time constraints can be assigned to the duration of the activity. Data flow between activities is accomplished by assigning data from messages to state variables and vice-versa. Rule sets express complex conditions based on XPATH expressions that can be used to filter input messages to activities.

BPML provides comprehensive **transaction management** support for both coordinated and extended transactions. The coordinated transaction model is based on ACID requirements and relies on a two-phase commit protocol. The extended transaction model, on the other hand, deals with long-running transactions. Such transactions use short resource locking, as they can be formed by nested transactions. The extended model allows for transaction interleaving that occurs in complex multi-party collaborations.

Activity compensation can be performed for both coordinated and extended transactions. Aborting a transaction triggers any compensating activities within the same context, in reverse order.

Robust **exception handling** capabilities are provided. The techniques are similar to XLANG. Exceptions are propagated upwards to enclosing activities until caught. If a transaction with no exception handler raises an error, the transaction is aborted.

BPML offers abstract processes that are XML documents. An abstract process specifies aspects of their behavior relevant to the overall process model; it can omit information that is not pertinent to a particular set of participants. Therefore abstract processes resemble public process models in ebXML or XLANG and mappings between

these standards are simplified. As the **service interface** part of abstract processes is very similar to WSDL, a WSDL description of a Web service could be mapped to an abstract process.

BPML process participants can represent IT systems, applications or users within an organization, or external service providers. Binding information is left to the implementation.

BPML does not support *security and reliability semantics, non-repudiation semantics and agreements.*

4.8. Electronic Business XML (ebXML)

4.8.1. Overview of ebXML

Electronic Business XML (ebXML) is a horizontal standard, for use within vertical industries, developed by the initiatives of UN/CEFACT and OASIS. ebXML complementary specifications are almost mature, as most of them reached version. The complementary ebXML specifications can be grouped into categories such as Messaging Services, Registry and Repository, Process execution and management, Process definition, CPP generation, CPA generation, Business document definition etc. ebXML's purpose is to enable enterprises of any size, in any location to meet and conduct business trough XML-based message-exchange.

The promise of ebXML is to create a single global electronic market and its mission is stated as "*To provide an open XML-based infrastructure enabling the global use of electronic business information in an interoperable, secure and consistent manner by all parties*" (ebxml, 2002). ebXML can address the needs of medium- and small-sized enterprises by its modular infrastructure suitable for inter-enterprise and intra-enterprise business process integration. The major function of ebXML within this context is automating loosely coupled document centric business collaborations. Web services and

ebXML can be used in combination to realize the goal of industrial-strength services; they should not be represented as technologies replacing the other.

The main ebXML operations are modeling business process and information, mapping the model to XML schemas, and defining requirements for applications that process the documents and exchange them among trading partners. Partners can find each other using a registry, define contracts, and exchange XML messages for their business operations over the Internet.

4.8.2. ebXML Specification

The following features are a general list for the actual ebXML specification:

- ***Process Specification*** covers integration scenarios that span multiple Binary Collaborations or Multiparty Collaborations.
- ***Binary Collaboration*** covers integration scenarios that span multiple Business Transactions.
- ***Multiparty Collaboration*** covers integration scenarios that span multiple Binary Collaborations. These Collaborations can be assembled into a Multiparty Collaboration.
- ***Business Transaction*** defines explicit Request and Response message exchange sequences Business Action defines in particular the XML documents that are exchanged as part of the message.
- ***Collaboration Protocol Profile (CPP)*** specifies the capabilities of a given company. A business organization can describe its profile, which includes the business processes it supports, its messages exchanges, its transport protocols and its role in a business process.
- ***Collaboration Protocol Agreement (CPA)*** specifies the contract between two Business Partners as the intersection of two CPPs. It acts as a business contract, a trading agreement, between business participants. Such a contract includes terms such as the specified messaging protocol, transport protocol, etc., however no information about Quality-of-Service (QoS) constraints, price/penalty is provided.

- ***Registries*** are global systems accessible by all via an API using the ebXML Messaging Service to publish and discover CPPs. The ebXML Registry artifacts are business process and information metamodels, business library, core library, collaboration protocol profiles, list of scenarios, messaging constraints and security constraints. A business partner publishes its profile with the ebXML Repository and enables other business organizations to access the profile.
- ***Messaging Service*** defines the communication and transport mechanisms for exchanging messages. The ebXML messaging specification is based on SOAP with Attachments and does not use WSDL, however it has several important features such as security, guaranteed messaging, and compliance with business process interaction patterns.

Web services and ebXML offer similar operations provided by electronic data interchange (EDI), except that they address requirements for the Internet. In addition, ebXML includes document-oriented interactions and offers standards such as industry-specific XML "vocabularies".

4.9. Standardization Groups

4.9.1. W3C Web Services Choreography Activity Group

The W3C Web Service Choreography Activity Group, which was set up in January 2003, works on existing specifications in order to determine capabilities, limitations, similarities and advantages. Clearly, an industry-wide Web service orchestration standard would streamline business-to-business application integration.

4.9.2. OASIS Web Services Business Process Execution Language Technical Committee (WSBPEL)

The purpose of the Web Services Business Process Execution Language Technical Committee (WSBPEL) is to continue work on the business process language published in the Business Process Execution Language for Web Services (BPEL4WS) specification in August 2002. Continuing the approach and design used in BPEL4WS, the work of the

BPEL TC will focus on specifying the common concepts for a business process execution language which form the necessary technical foundation for multiple usage patterns including both the process interface descriptions required for business protocols and executable process models (OASIS, 2003).

4.10. Comparative Study Results

This section makes an overview of business process description languages introduced so far. Table 4.4 is an incomplete checklist outlining features provided by each language. Thus drawing early conclusions and making estimations about the specifications will be simplified. The selected criteria is based on a variety of comparison efforts such as (Dubray, 2003), (Haberl, 2003), (Wohed *et al.*, 2003a) and (Peltz, 2003). The work presented in (Wohed *et al.*, 2003a) and (van der Aalst *et al.*, 2003b) is a comparative study that makes use of workflow and communication patterns to analyze BPEL4WS and BPML/WSCI respectively.

Table 4.4. Business process description languages - checklist

Specification	BPEL4WS	BPML	WSCI	WSFL	XLANG
Data flow	XML	XML	XML	XML	XML
Control flow	Block structured	Block structured		Transitions (Graph-model)	Block structured
Message flow	WS	WS & Global Model	WS & Global Model	WS & Global Model	WS & Contracts
Transaction -Txn Long-Running-Txn	+	+	+	-	+
Roles	+	+	+	+	+
Lifecycle	+	+	+	+	+
Endpoint properties	-	-	-	-	-
Graphical Notation	-	-	-	-	-
PROCESS Concepts					
Aggregation	+	+	+	+	+
Branching	+	+	+	+	+
Loops	+	+	+	+	+
Time	+	+	+	-	+
Exceptions	+	+	+	+	+
Internal/External	+	+	+	+	-
Multi Choice	+	-	-	+	-
Sequence	+	+	+	+	+

4.10.1. Relation and Relevance of Existing Specifications

The following results are based on Table 4.4 and make up a brief comparative summary of the specification sections.

The general critic made about **BPEL4WS** is its complexity. BPEL4WS offers a diverse range of language constructs, as it is based on WSFL and XLANG. There are many overlapping language elements, due to the convergence of two languages. Conclusively, designing flow and composition can become complicated. Another negative remark is being made about unclear BPEL4WS semantics by (Wohed *et al.*, 2003). For example, the high-level scope language element can be misinterpreted by different developers.

BPEL4WS applications are *process-centric.* The structure of a BPELWS application consists of a top and bottom layers, where the former is written in BPEL and represents the flow logic of the application and the latter represents the function logic of the application. An abstraction of layers simplifies application changes, since the business process and the invoked Web services are not tightly coupled to each other.

Applications written in BPEL have another major advantage over conventional approaches, as they allow tailoring the ready application to the needs and circumstances of a particular environment without touching the application itself. Aside from layer separation, partner definitions are also separated from actual process participants; instead only port types and operations for different partners are specified.

A significant difference of BPEL4WS is its support communication modeling language constructs.

WS-Transaction uses WS-Coordination to extend BPEL4WS to provide a context for transactional agreements between services. Different agreements may be described in an attempt to achieve consistent, desirable behavior while respecting service autonomy. WS-Transaction also specifies protocols for coordinating distributed atomic transactions. A

future extension of BPEL may support distributed atomic transactions consisting of activities of a single process or even multiple processes.

As mentioned before in the BPEL4WS section, process engines can interpret executable BPEL4WS scripts so that modeling business activities is sufficient to implement business processes.

The **BPML** specification provides language constructs and role/partner elements similar to BPEL4WS. Support for basic and structured activities is also available. BPML provides task scheduling constructs.

The compensation logic for transactions using scoping features is provided in BPEL4WS. But BPML provides the ability to nest processes and transactions, a feature that BPEL4WS currently does not offer. Finally, a robust exception handling mechanism is available within BPML, following many of the constructs in XLANG. Timeout constraints can also be specified for specific activities defined within the process.

A key aspect of **WSCI** is that it only describes the observable or visible behavior between web services. WSCI does not address the definition of executable business processes as defined by BPEL4WS. Furthermore, a single WSCI document only describes one partner's participation in a message exchange.

4.10.2. Convergence of Specifications - Concluding Remarks

Software vendors with "candidate" specifications still strive for competition to take the lead in the emerging Web Service Orchestration area, so there are various specification offerings. It is uncertain yet, which specification finalist will become the winner of this competition. On the contrary, a convergence of standards in the Web Service Orchestration arena could become a breakthrough for the evolution of Web service standards.

A straightforward solution would be to merge existing specifications so as to form a new standard that has complementary features of each standard. To single out only one of

these specifications as an industry standard would also be a reasonable approach. In any case, a unified specification adds momentum to standardization efforts.

Standardized business process management (BPM) solutions are the key to business-to-business application interaction and such advancement would accelerate widespread adoption of Web services. In addition, convergence would increase the tool choices for developers/users. Complementary Web service BPM technologies such as modeling tools, execution engines etc. could be based on a single standard, which would prevent custom-coded and vendor-specific incompatible solutions. Moreover, if complex business issues were addressed through standards, implementation of real business scenarios could spread out and give companies the opportunity to evaluate potential Web service benefits. Conclusively, software vendors could also observe these case studies to determine further requirements for Web service BPM solutions. With a unified orchestration and choreography standard, businesses can communicate with each other without any transformations, even though they speak different languages in private. Through communication more complex functionality and value-added services can be achieved.

5. FINANCIAL WEB SERVICE USE CASE

5.1. Web Services in the Finance Industry

5.1.1. Financial Service Challenges

Companies in the finance sector, especially banks, provide a diverse range of services and products to various customer groups, such as Internet banking, credit card products, ATM services, phone banking, call center services to clients and merchants. Multi-channel services, new products offerings or ever-changing business operations require flexible and interoperable system architectures or steady support for reorganization of IT infrastructure, where the latter choice is a routine challenge and not preferable in the competitive financial market. Certainly, not all of the software applications and implementations are developed in-house and companies have to spend a vast amount of time for system integration and interoperability.

IT investment for the financial services sector is worldwide very high and companies make use of latest technology. Many developers within the IT department of a financial institution are already familiar with Internet standards. Most companies already use the Internet to deliver their services and products online and utilize Internet standards, such as TCP/IP, HTTP, HTML, XML, J2EE, .Net – the building blocks and principal players of Web services. Due to this fact, the technology adoption life cycle for Web services can take less time in an already Web-enabled environment. Therefore a movement towards Web service methodologies is not a decision that gives rise to massive reorganization of the technological architecture of such a company.

5.1.2. Potential Web Service Benefits

Web services provide several bridging and integration techniques for designing and implementing loosely coupled and compatible applications on distributed software domains. In case of financial web services, the deployment and development time and cost

of multi-channel services could be reduced by a reasonable extent. Making key financial business processes available as Web services can simplify application interaction and integration for companies that make use of distributed computing.

Web services are especially useful for enterprises that access internal and external information and application services. Generally, there are redundant application services that exist at two or more systems. If a new application must be developed due to a business need, it must also facilitate communication with application services residing in remote systems.

Financial institutions rely heavily on information sharing and the Web provides distinguished opportunities for data presentation and transfer. Applications based on Web services' standards, make internal data more accessible inside and beyond company borders. With a technology like Web services, companies also have a chance for internal and external business processing and transaction coordination. Business process integration of partners is another possibility and the flexible technological infrastructure of Web services simplifies the "making together business" issue.

Turning to the merits of Web services' strategic concept; companies can concentrate on real business instead of ever-changing IT requirements. Consequently, we are faced with a more business-centric approach than a technology-centric approach. This is a key requirement for a company to adapt itself to the rapidly changing marketplace. To get the most out of this promising technology, a service-oriented approach in conducting business is a must. Interestingly, "focus on services" is not an obstacle as the principal concern of a company, in our case a financial institution, is to deliver services to its customers.

Web services should not just be evaluated within the scope of a cost-efficient technology, since it can also facilitate value-added services. Companies in the finance sector aim to provide multiple products and services over various distribution channels to their customers. The current state of such operations is that most channels require excessive application changes. Taking platform-independent Web services into consideration, the development of a single application, with an interface based on Web

service standards, is sufficient to offer a product or service through multi distribution-channels. Moreover, client back-end operation and presentation logic can be customized for each channel separately and independent of the Web service's implementation.

Even companies with limited IT resources, which offer financial services, can use up-to-date business models with outsourcing Web service based technologies. Due to such a shift in doing business, there would be no longer a demand for high-cost IT investments or very specialized IT skills and operational cost would decrease.

Service-oriented integration is the abstraction of behavior as well as information between information systems. Web services are an example of this. EAI is the unrestricted sharing of information between multiple systems intra-company. B2B integration is the unrestricted sharing of information between multiple systems inter-company.

5.2. Financial Web Service Use Case – Project Scope

The motive of this section is to identify the challenges of Web services adoption for real business use cases in a distributed computing environment. Therefore a typical distributed computing example from the Finance Sector has been selected to produce an experimental Web service scenario and solution.

Comparing the wealth of Web service specifications to the small number of real business examples, triggered the following idea: "*For a so-called warm-up in Web service implementation, instead of awaiting finalized standardization studies, a staged development method should be put into practice* ".

5.3. Financial Web Service Use Case - Business Perspective

5.3.1. The Big Picture

The overall scenario includes an enterprise that wants to implement a business process for processing purchase orders online using a set of Web Services. Thus first the following steps must be taken (Leymann, 2001):

- Identification of the business process (e.g. credit history checking operation, process order, reject order, ship goods, refund)
- Specifying business rules for sequencing of process steps.
- Specifying the flow of information between the process steps

After the requirements have been identified, the enterprise would search for Web services that already offer generic operations such as credit card checking service, supplier service, shipping service, etc.

The enterprise-side of the scenario is a bottom-up analysis of the big picture. For the actual implementation, the credit-card checking service has been selected. This is a straightforward service that is already provided, however that uses proprietary protocols. The purpose is to simplify implementation and provide a "plug-in" service that enables the service requestor to increase the functionality of their systems with minimal systematic changes. A credit-card authorization service offered by a financial institution will be transformed into a Web service that is entitled "PaymentAuthorization" in the context of the presented use case. Merchants, who want to start-up E-Commerce operations, acquire a Virtual POS software module from the bank, which they use for online authorization requests and payment charging procedures.

An online payment-authorization service is a real-time process, which runs in a distributed computing environment and requires besides full system integration also security considerations on various platforms. Therefore, it is a profound use case to examine the adoption of Web service technologies for financial services. An entire

authorization process has to be completed within seconds (max. 1-2 seconds). So the complete process life cycle cannot be described as a long- running transaction. However, as soon as the demand for sophisticated internal/external integration solutions with the service arises, describing business processes that interact with the service will play a key role. Thus the staged development approach will provide the flexibility to equip the service new features.

5.3.2. Existing Payment Authorization Service

A merchant sends a payment authorization request to company XYZ that acts as a mediator throughout the payment process. Company XYZ also provides IT support and ASP-like services to merchants that do E-Commerce. The actual application architecture and data flow is displayed in Figure 5.1.

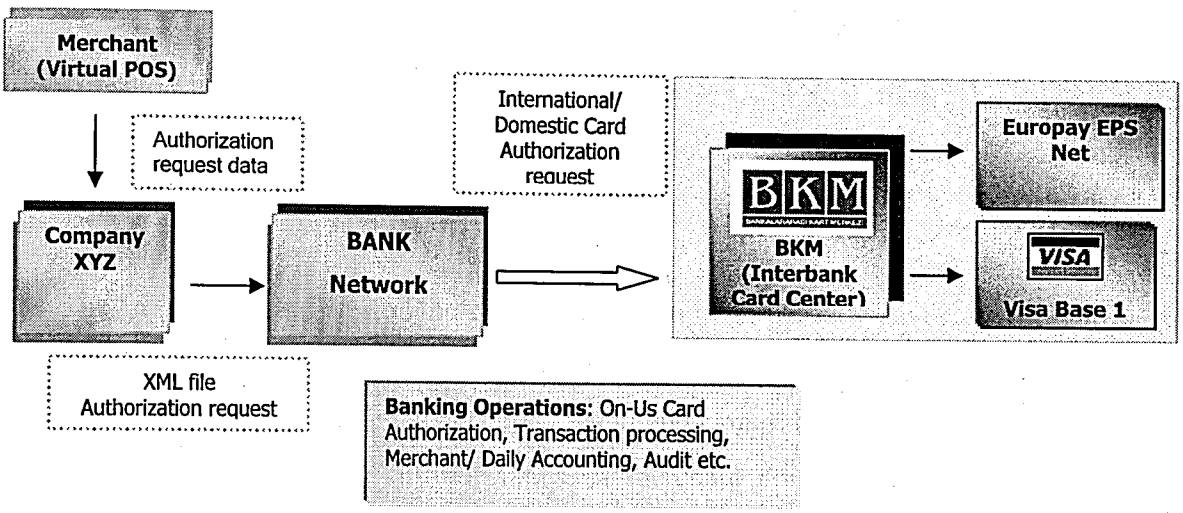

Figure 5.1. Payment authorization service

- Company XYZ receives the authorization request from the merchant and converts it into XML data in a certain format based on the specifications identified by the bank.
- Company XYZ forwards the XML file to the bank's payment application via HTTPS.

- The XML data received is parsed into an actual authorization request and the authorization process is invoked as a database procedure already existing in the bank's payment system.
- According to the card-type (on-us, domestic or international), authorization request processing differs:
    - On-Us cards are handled within the existing authorization process and a response is produced based on the customer's records.
    - An authorization request for a domestic card is sent via the Interbank Card Center (BKM) to the card-issuing bank and a response is received.
    - Authorization requests for international cards are transferred via BKM to VISA/EUROPAY, where these are sent to the card-issuing bank. In Turkey, BKM is the Interbank Card Center that acts as a central operation site for domestic clearing and settlement of daily debit/credit card transactions and domestic/international authorization switching and message routing between member banks or Visa/Europay.
- An authorization response is returned to Company XYZ as XML data.
- Company XYZ forwards the authorization response to the merchant in a format identified by the merchant.

The principal drawback of the original application is the extensive use of XML-parsing, which can become a significant performance penalty. Another disadvantage is that the application interface is not applicable for complete reusability across internal and external services without any customizations. A merchant doing E-commerce uses the described application and call methods, whereas an internal authorization process request received through an internal service calls the application directly with specified parameters instead of a XML-message that requires parsing. These method calls need to be unified for internal and external application requests.

5.3.3. Sample Scenario for a Payment Authorization Web Service

The plan is to wrap the online credit card validation and real-time payment authorization service with Web service features. A wide range of products automating Web service standards coding exist.

Based on the sample scenario, an authorization request, sent directly by a merchant to the bank, contains specific parameters such as *terminal-id, transaction- amount, card-number etc.* The Web service scenario is displayed in Figure 5.2.

- A merchant with a virtual POS installation, which communicates via HTTPS, sends an authorization request message to the Payment-Authorization Web service using SOAP.
- After the Payment Authorization Web service has been invoked, field-level control checking is performed. According to the card-type (on-us, domestic or international) the authorization request processing differs.
- The response for an authorization request is returned through the Payment-Authorization Web service.
- The merchant displays, prints or logs the authorization response information as needed.

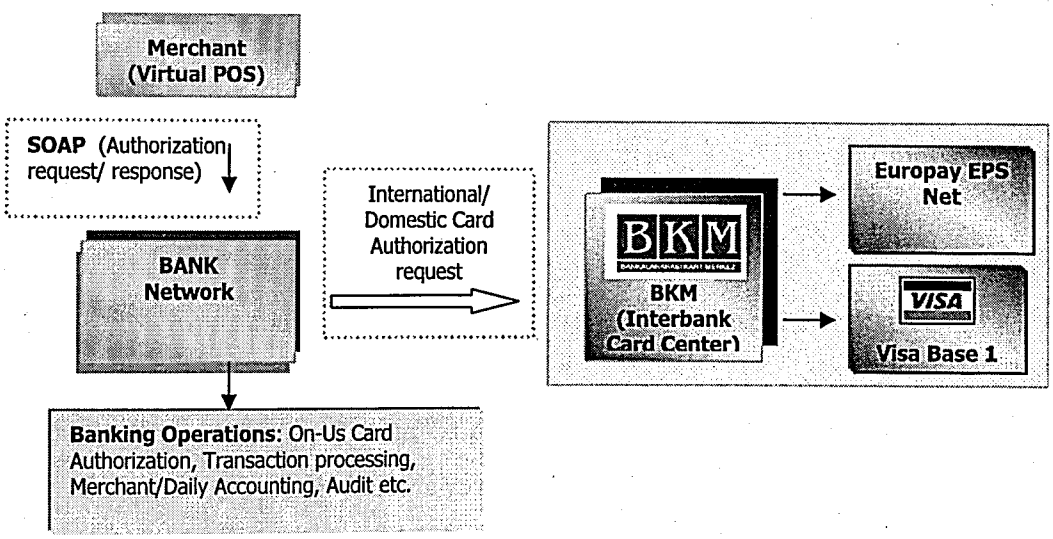

Figure 5.2. Payment authorization Web service

Merchants can create customized user interfaces for user input and screens displaying response information. The client application's presentation logic is not dependent on the authorization Web service.

The Payment-Authorization Web service describes its own functionality due to its standards based WSDL file. It can easily be integrated with internal applications, such as applications that handle authorization requests from branches or mobile payment applications, without the development of several client interfaces for varying service calls. With further modifications, this Web service can also promise dynamic interaction and automatic self- integration features.

Due to security considerations, merchant-bank communication is established through HTTPS. A further security aspect is the control of clients' IP addresses during Web service invocation. The current experimental system architecture does only permit access to On-Us merchants for the Payment-Authorization Web service, because the system provides merchant accounting, card authorization and settlement operations for the entire payment process. If the system would also act as a router to redirect the authorization request messages of Not–On-Us merchants' to their actual financial institution, IP address controls would become unnecessary. Moreover, the Payment Authorization Web service would be published in a UDDI registry and turn into a single-source processing application for merchants. A single-source payment processing service has various business benefits. Due to the fact that a combined payment interface simplifies real-time transaction processing for the merchant, customer satisfaction is achieved - the essential goal of all companies competing in the finance market. Single-source payment processing and message-routing issues might be taken into consideration as future enhancements.

5.4. Financial Web Service Use Case - Development Perspective

As the "PaymentAuthorization" is provided as a financial Web service, the financial institution has the role of the ***service provider***. The typical development steps from the service provider view are (Cerami, 2002) explained in detail in the following sections.

Development of the core functionality of the service includes development and testing of the Web service implementation, the definition of the service interface description and the definition of the service implementation description. Web service implementations can be provided by creating new Web Services, transforming existing applications into Web Services, and composing new Web Services from other Web Services and applications.

For the use case, the development phase involves the connection to databases, Enterprise JavaBeans (EJBs) or legacy applications. *The authorization application is a stored database procedure hosted on the back-end switching system of the financial institution.*

Development of a Web service wrapper for the original application wraps existing functionality into a larger framework. The actual authorization operation of the application has been decomposed into a service implemented in Java; this turns the application into service offering.

The initial service handles one operation that is used to verify, authorize and return a status message. Basically, credit card information input is used to produce an output message. The service is bound to a concrete implementation provided by a local Java class.

The use case makes use of SOAP; therefore a WSDL file must be created. Creating the service description file named "PaymentAuthorization.wsdl" forms the *provision of the service description.*

WSDL types that are part of the service are mapped to local Java classes. Operations within the port types are mapped to methods in the Java class that provides the service; currently only a mapping for the "PaymentAuthorization" service exists.

Deployment of service includes the publication of the service interface and service implementation definition to a service requestor or service registry and deployment of the executables for the Web service into an execution environment.

Since the application already runs on an application server, the server can be enhanced with embedded Web service features. Otherwise, another standalone server must be used for deployment.

If the Web service is fully deployed, it becomes operational and network-accessible from the service provider. Find and bind operations can be performed. During invocation, messages intended for this service are de-serialized into Java types and the appropriate method of the provider class is invoked with the parameters. Conversely, the output resulting from the invocation of the Java method is serialized to an XML message of the appropriate type.

Publishing a service can be done through a global registry, e.g. UDDI. UDDI discovery operations use XML. Therefore service requestor and provider can implement services in different programming languages. In case that a common language is used, service discovery can also be done through language semantics instead of XML and speed up the discovery process. The UDDI registry includes many abstract data and this can also become a drawback for performance. As the current design of the service assumes point-to-point communication, UDDI discovery are not necessary anyway and the last can be left out from the development stages.

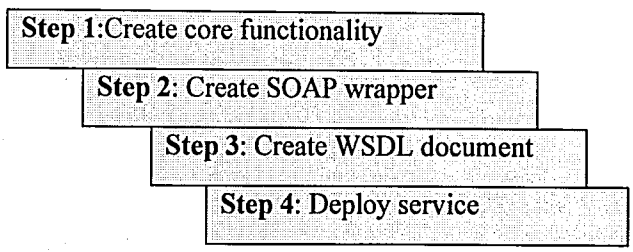

Figure 5.3. Service provider development steps

Figure 5.3 is a snaphot of the development steps for the use case is displayed. The development steps match with the development lifecycle phases explained in the Web Services Development Concepts paper (Brittenham and Davis, 2003).

5.5. Financial Web Service Use Case - Technology Perspective

5.5.1. Web Service Technologies

The landscape for Web services includes a range of technologies, spanning both the service requestor's environment and the service provider's environment. Figure 5.4 (Aldrich, 2002) illustrates a sample technology environment. In the figure, the requestor application runs in a .NET environment, and the provider's application runs in a Java environment.

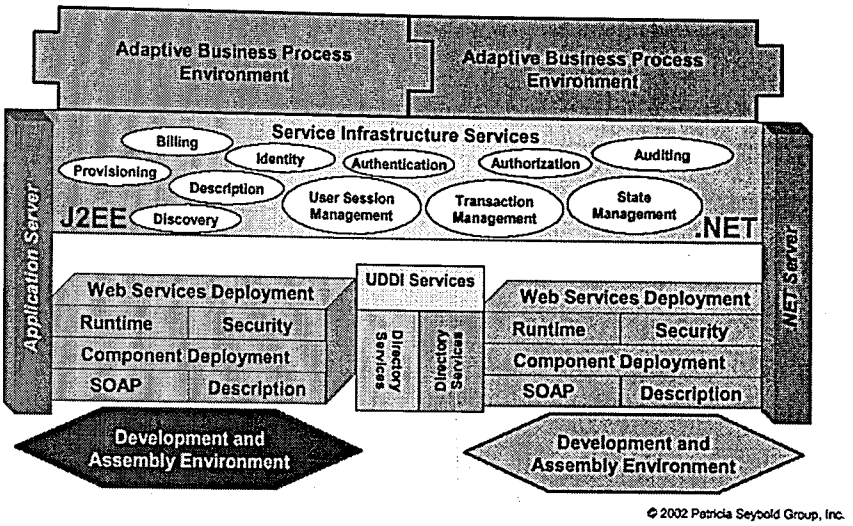

Figure 5.4. Web service technology landscape

There are a wide range of tools automate Web service development and integration concepts. Due to the evolving state of Web services, many tools release constantly new versions. Furthermore to catch up in the race, many products are also published without exhaustive controls, beta-phases and documentation. Especially developer tools that are offered for a limited time do not extend licenses, as already a new version of the tool is made available. Therefore a routine download and setup operation is inevitable. Some software vendors offer extensive mail support, however some vendors make developers struggle until the final decision to switch to another tool is made.

5.5.2. Tools Overview

Table 5.1 lists the key tools that have been used to develop, deploy and test Web service samples. The list provides only a general overview; further details for some tools are included in the following sections.

Table 5.1. Software development tools

Tool	Software Product
Actual Development Environment	
RDBMS	Oracle 9i
Application Server	BEA - WebLogic
	TomCat
Java IDE	Eclipse for Java
Web Service Tools	
Web Service Orchestration	Collaxa Orchestration Server
	BindStudio WSO
BPEL4WS tools	IBM BPEL4J Editor
WSCI tools	Sun WSCI Editor
Web Service Development	Systinet Wasp Developer – Eclipse plugin CapeClear – CapeConnect Studio
Web Service Server (Deployment)	Systinet - Wasp Server for Java Cape Clear – CapeConnect

The financial web service has been created by mixing together existing applications with Web service enabled interfaces. During the development and transformation phase, many Web service software solutions had to be evaluated. Evaluation was not only a personal or project-based decision; it was more like a must, as many solutions did not fulfill the requirements of the existing application. Although the payment authorization application itself has a basic request-response structure, using the underlying infrastructure without any changes forced us to consider several Web service tools.

5.5.3. Systinet WASP

Systinet releases a suite of products, which form its Web Applications and Services Platform (WASP).

Wasp Developer is a free plug-in that extends the Eclipse IDE to allow for web services application development, testing, and interface creation. WASP Server for Java is a fully featured runtime environment for deploying web services applications.

The diagram in Figure 5.5 shows the runtime operations of WASP (Aldrich, 2002). This diagram depicts WASP Server client request execution. Execution steps are:

1. Locating the WSDL for the Web service
2. Creating the client proxy for the service
3. Checking secure identity
4. Invoking and managing application processors.

In the diagram, legacy application services are colored black, infrastructure box is colored white, while WASP functions are in dark and light gray.

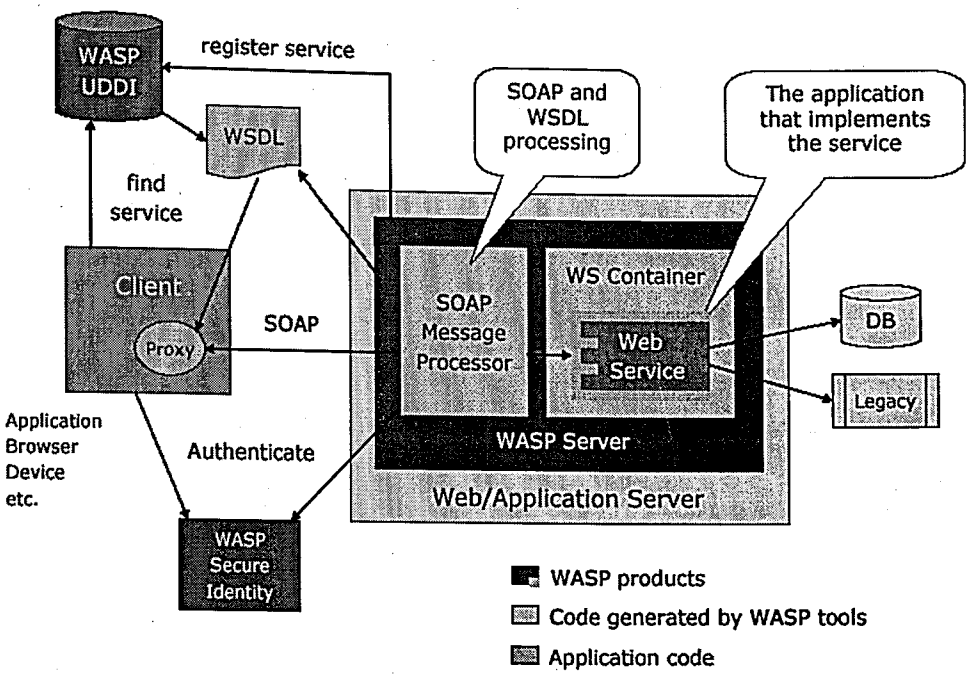

Figure 5.5. WASP runtime operations

To get hands-on-experience, making basic method invocation applications as a service requestor works out flawless. Though being the service provider and offering a service with database connections or already configured application servers requires detailed analysis. Conclusively, the overall development phase does not resemble the "plug and play" assertion, even with most Web service tools that already automate many transformations. But the notion of using Internet standards simplifies the adoption of the Web service development concept. A web service client developed for the well-known Google API is illustrated in Figure 5.6. The "GoogleClient" looks up the WSDL location and retrieves service descriptions. The query string is "BPEL4WS". If the client is executed a maximum of 10 search results is displayed. The example is not related to our financial use case, however displays the nature of a basic Web service client and typical request/response Web service.

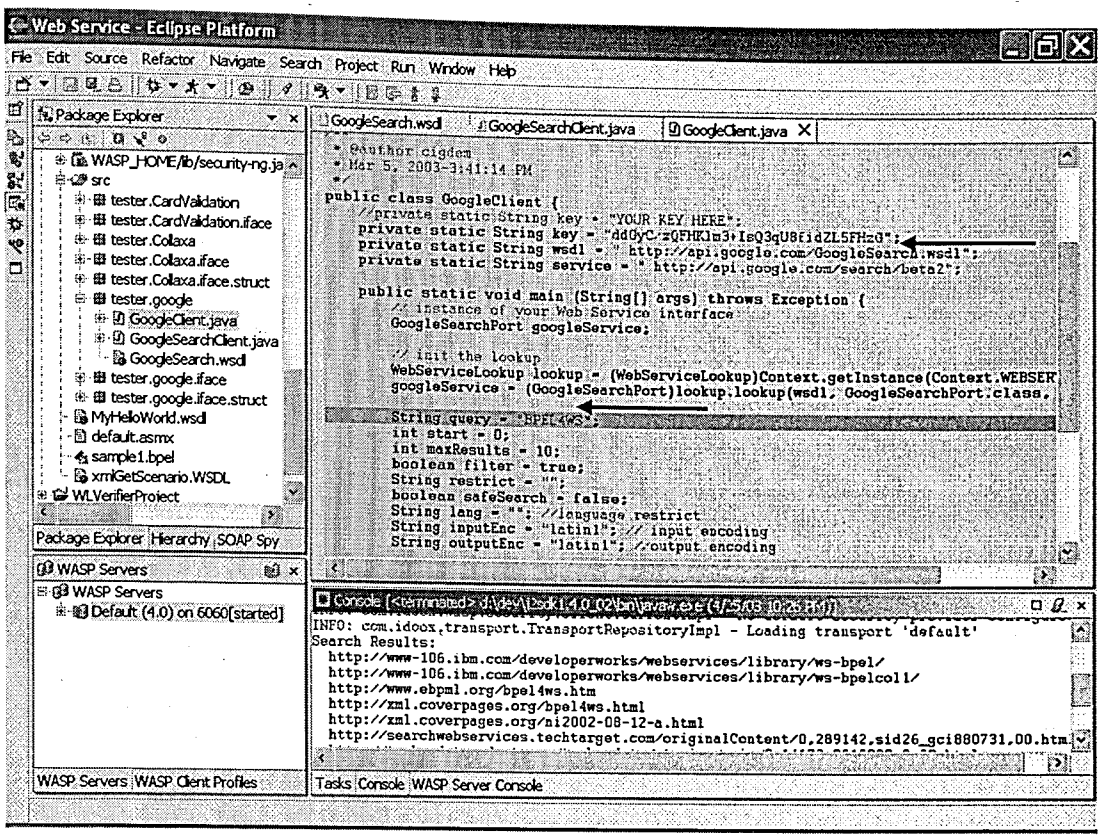

Figure 5.6. Google API Web service client

As mentioned before, the original "Authorization" application retrieves information from an Oracle database so that application Server (BEA Weblogic) database naming and lookup operations must be used to execute database operations. The existing Java application has been transformed successfully into a Web Service with Wasp Developer, however the deployment and service invocation process did not support BEA Weblogic specific operations. The resulting error is displayed in Figure 5.7.

When Application server operations fail database lookup cannot be performed. Therefore the same deployment scenario has been created on WaspServer, which can be embedded into specific application servers. WSDL documents and web service interfaces have been created with Wasp Developer, however deployment worked only with a remote standalone Wasp Server. Service deployment within the Eclipse environment is depicted in Figure 5.8.

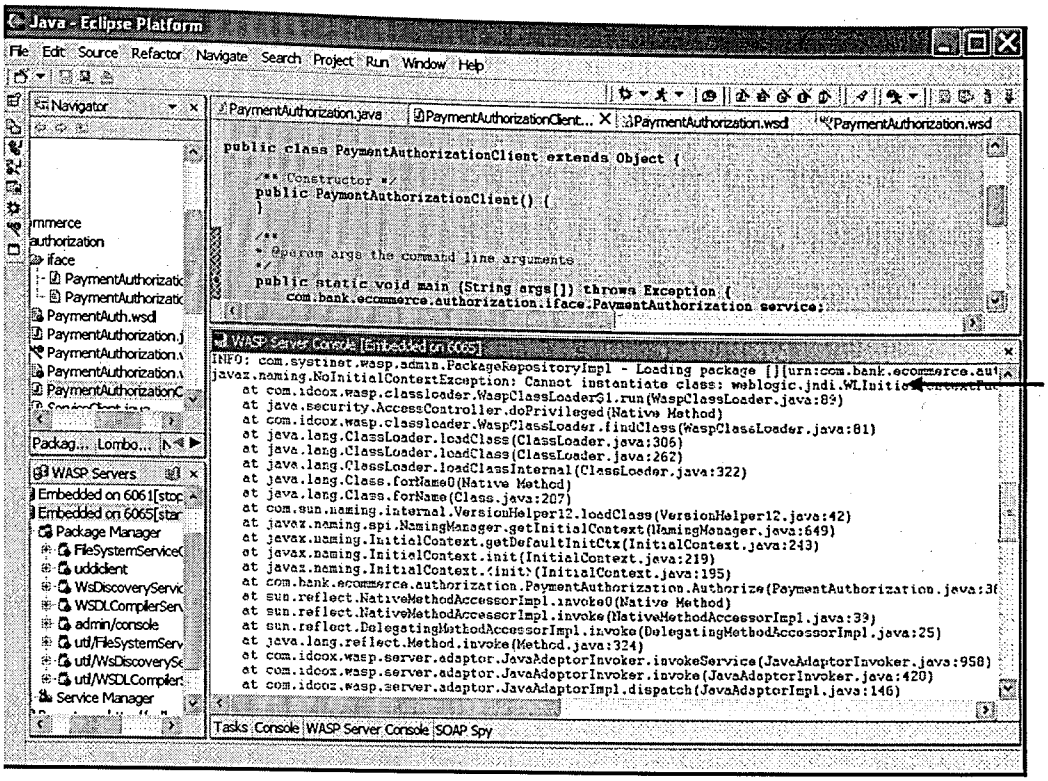

Figure 5.7. Wasp Developer – Application Server error

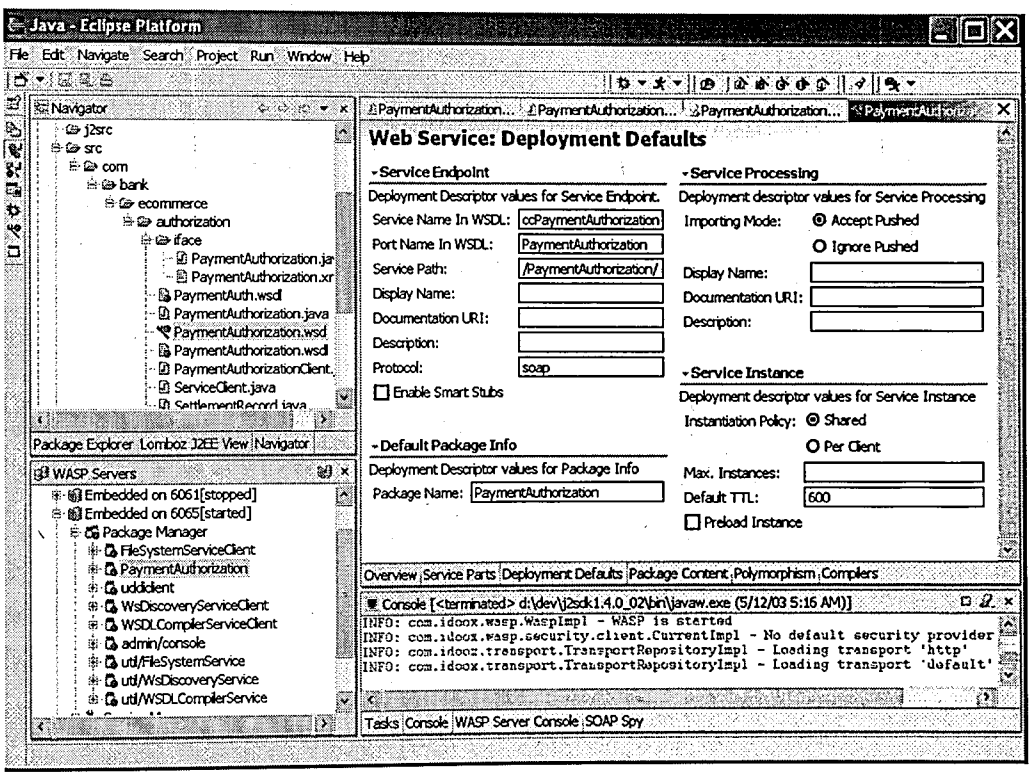

Figure 5.8. Wasp Developer/ Server – Service deployment

Many Web service tools are made up of an automated service development environment and a deployment server such as Systinet - WaspServer (Systinet, 2002) or CapeClear Studio (CapeClear, 2002). Deployment servers can be used as standalone servers or ported servers on an application server. Server porting operations can become tedious and are not based on straightforwards settings. Another important issue is that *standalone* Web service deployment servers reserve a certain port number and specific resources, so the developer may experience performance problems during implementation and testing.

Figure 5.9 shows the Web service client for the "PaymentAuthorization" service. Return values, exceptions and system values are displayed in the Wasp Server console as seen in the picture below.

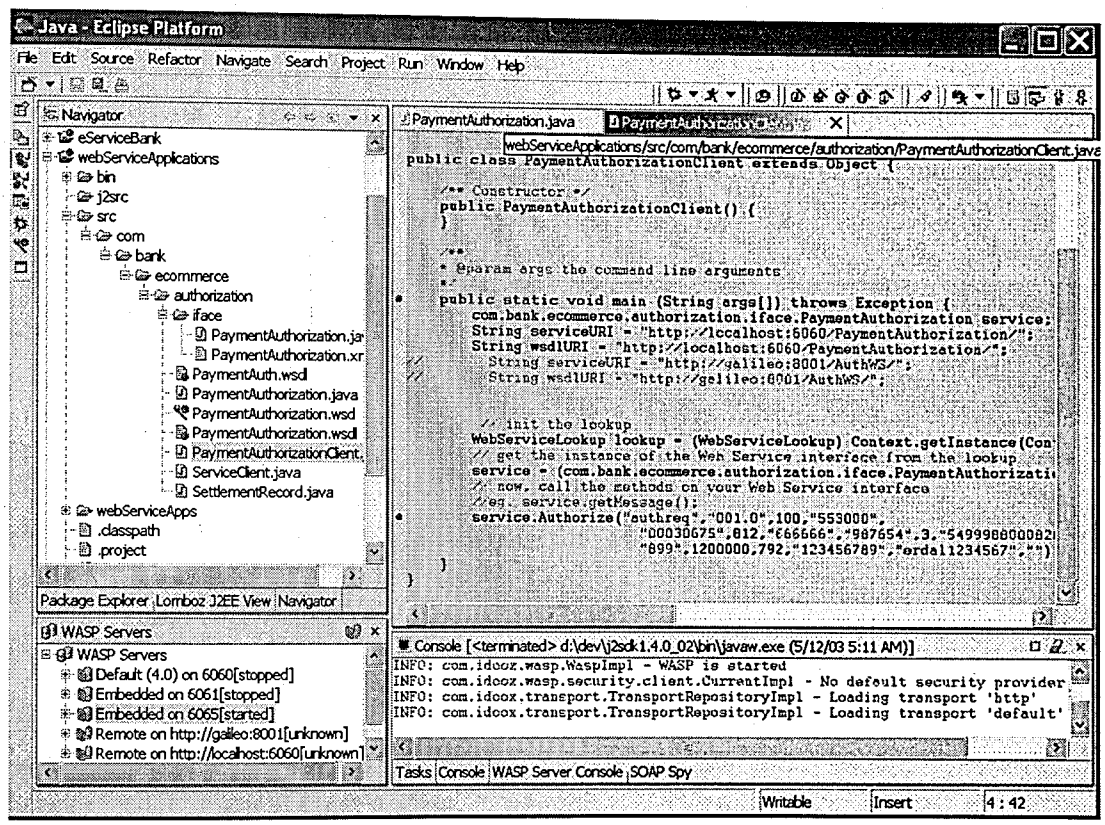

Figure 5.9. Systinet Wasp Developer – use case client

5.5.4. Cape Clear Studio

Cape Clear 4 Studio provides an integrated project environment for the design, development, integration, testing and deployment of Web Services (CapeClear, 2002). The project environment offers various features such as:

- WSDL editor to create and modify WSDL specifications.
- Automated WSDL generator for existing Java/J2EE/CORBA/COBOL components.
- Web service implementation assistants.
- Web service deployment and configuration wizards.
- Web service testing options and test-client generation

Web service usage should also accelerate application development speed. Therefore the "PaymentAuthorization" use case has also been deployed on the Cape Clear Server and clients have been generated for testing purposes. Figure 5.10 illustrates the general development environment and Figure 5.11 shows a test client with a JSP interface.

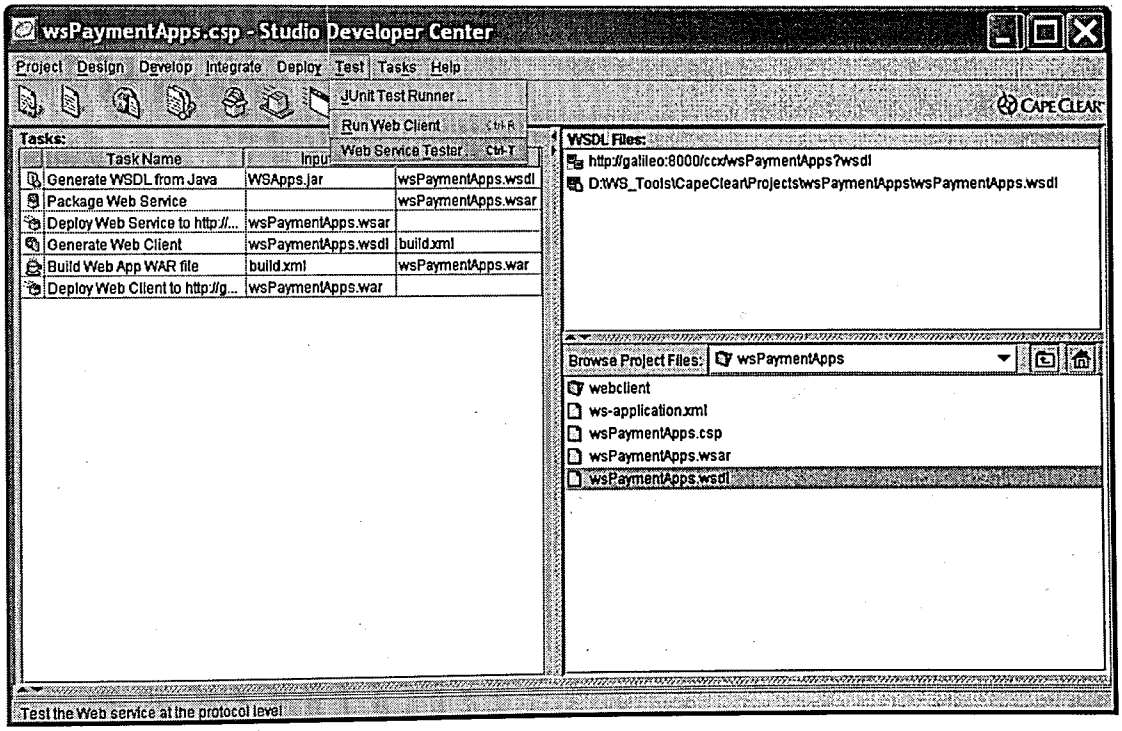

Figure 5.10. Cape Clear Studio – development environment

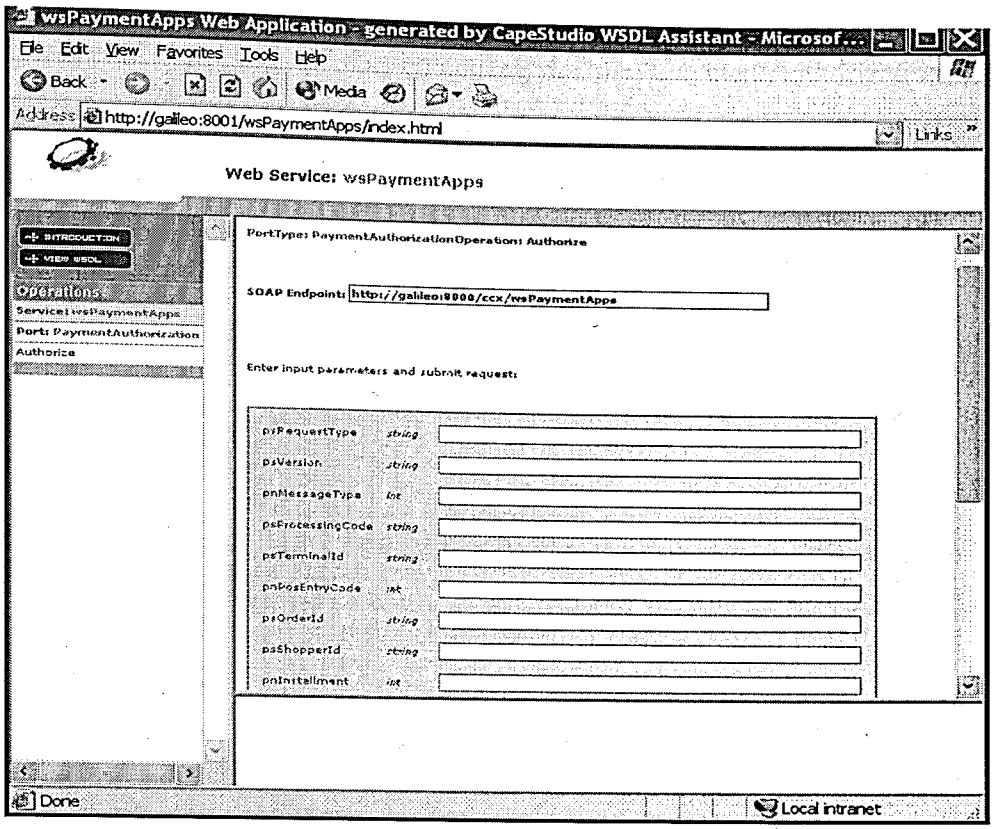

Figure 5.11. Cape Clear Studio – use case test client

The following server prompt window shown in Figure 5.12 lists output and trace messages produced by the invocation of the "PaymentAuthorization" service.

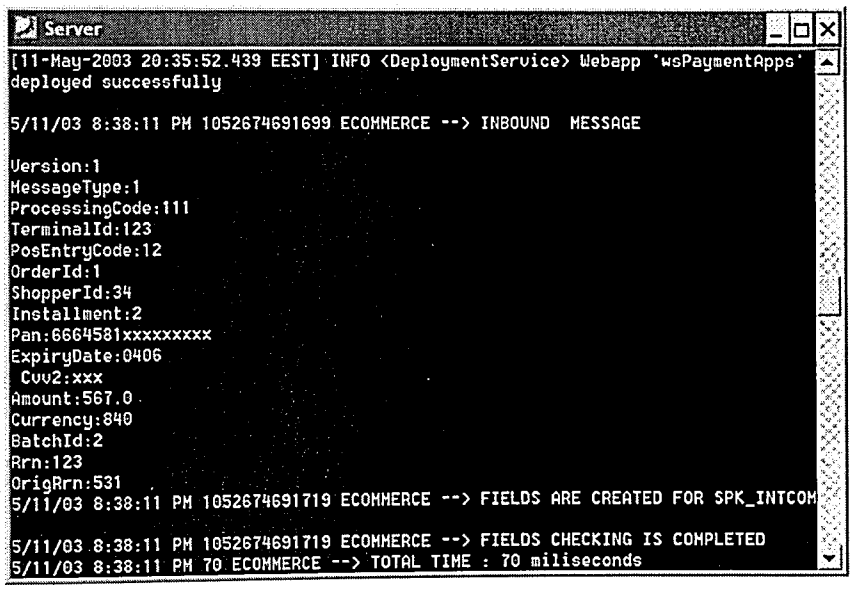

Figure 5.12. Cape Clear Studio – server output window

In Appendix A several WSDL documents for the "PaymentAuthorization" service are listed. Even though all Web service tools make use of "standard" Web service standards, the WSDL document created by Systinet Wasp Developer and Cape Clear Studio differs in description style and length. The Cape Clear Studio WSDL editor is displayed in Figure 5.13.

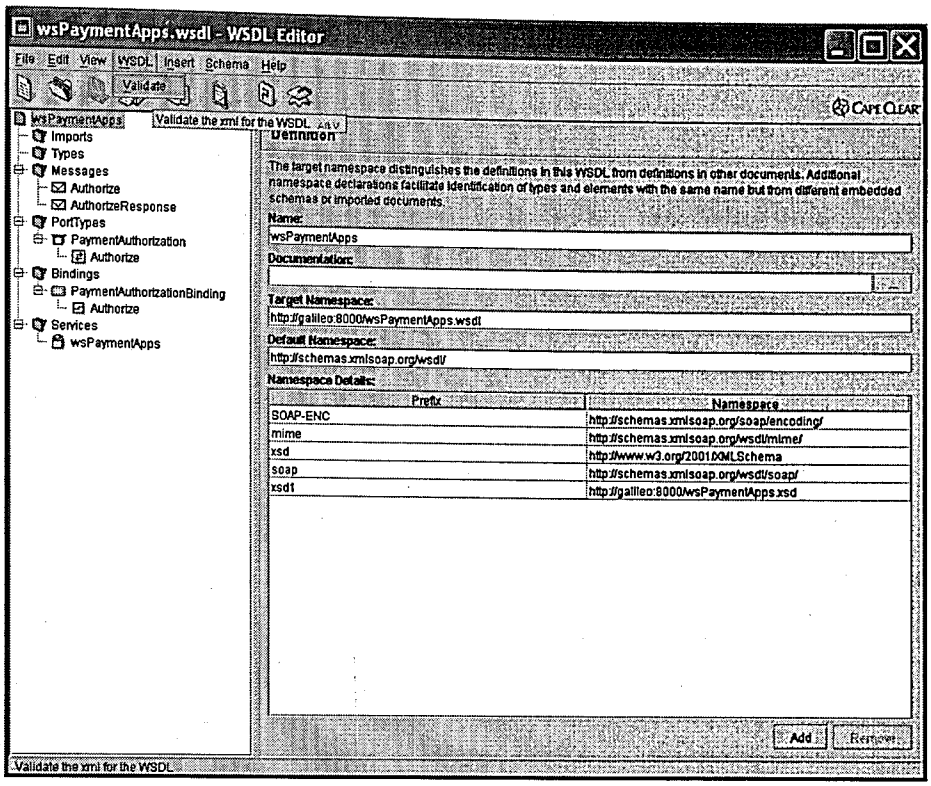

Figure 5.13. Cape Clear Studio WSDL editor

5.5.5. Collaxa BPEL Orchestration Server

BPEL Orchestration Server provides a run-time environment needed to execute, passivate and manage BPEL Scenarios. It coordinates multi-step workflows across asynchronous services and manages the integrity of nested compensating business transactions. It uses a database to passivate the context of BPEL Scenarios between each asynchronous interaction. (Collaxa, 2003a). Figure 5.14 outlines the overall infrastructure of the BPEL Orchestration Server (Collaxa, 2003a).

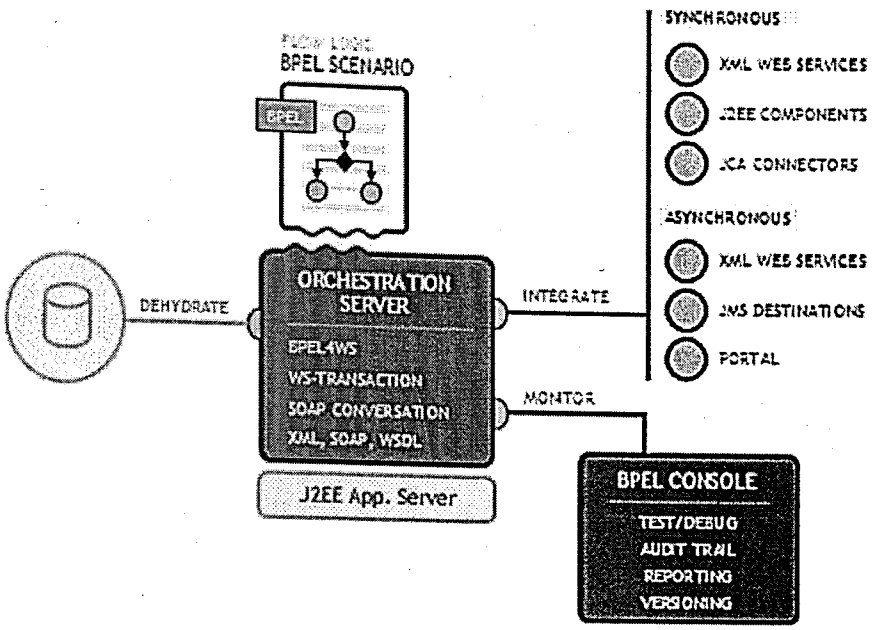

Figure 5.14. BPEL Orchestration Server infrastructure

The "PaymentAuthorization" web service is planned to be integrated into a complete business process flow and exposed as part of a service composition. Therefore the BPEL Orchestration Server seems to be a convenient tool for service deployment.

Collaxa provides several utilities such as "wsdlc" and "bpelc" for compiling and deploying scenarios, generating Web service proxies from WSDL files and generating business documents from .xsd files (Collaxa, 2003b).

The **"wsdlc"** utility can be used to create a Java proxy that represents the Web service interface defined in a WSDL file. If XML schema based business documents are used, "wsdlc" will automatically generate, compile and package Java proxy classes and create documentation for them. The ***BPEL Scenario*** is a programming abstraction for implementation of composite services, user tasks and JMS queues/topics into long-running, multi-step business flows. BPEL Scenario has built-in support of XML, SOAP, WSDL, BPEL4WS and WS-Transaction (Collaxa, 2002). Since the WSDL document for the "PaymentAuthorization" service has been created, implementing a web service client as a

BPEL Scenario is possible. Figure 5.15. shows how a two-way Web service proxy is created for our use case.

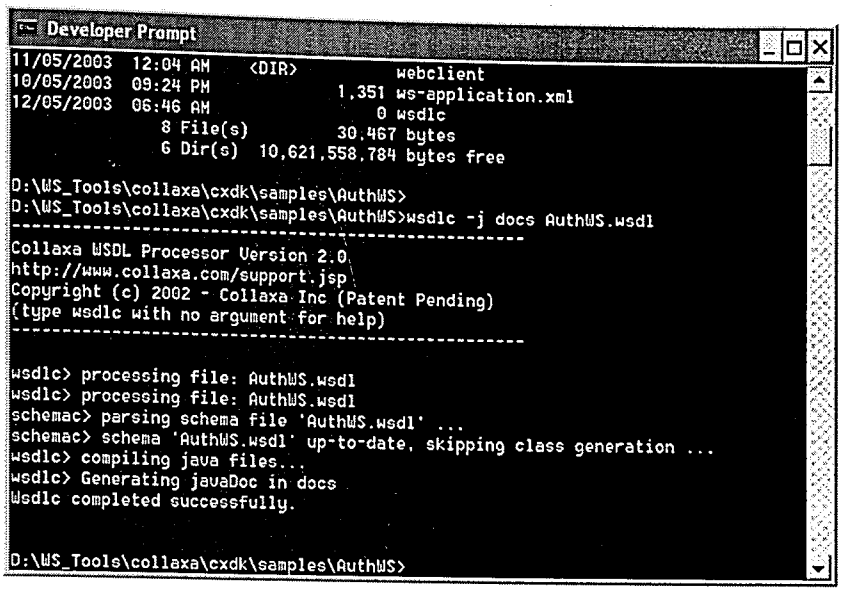

Figure 5.15. Collaxa "wsdlc" utility

The "wsdlc" utility can also create Java documentation for the web service and its related interfaces; this feature is depicted in Figure 5.16.

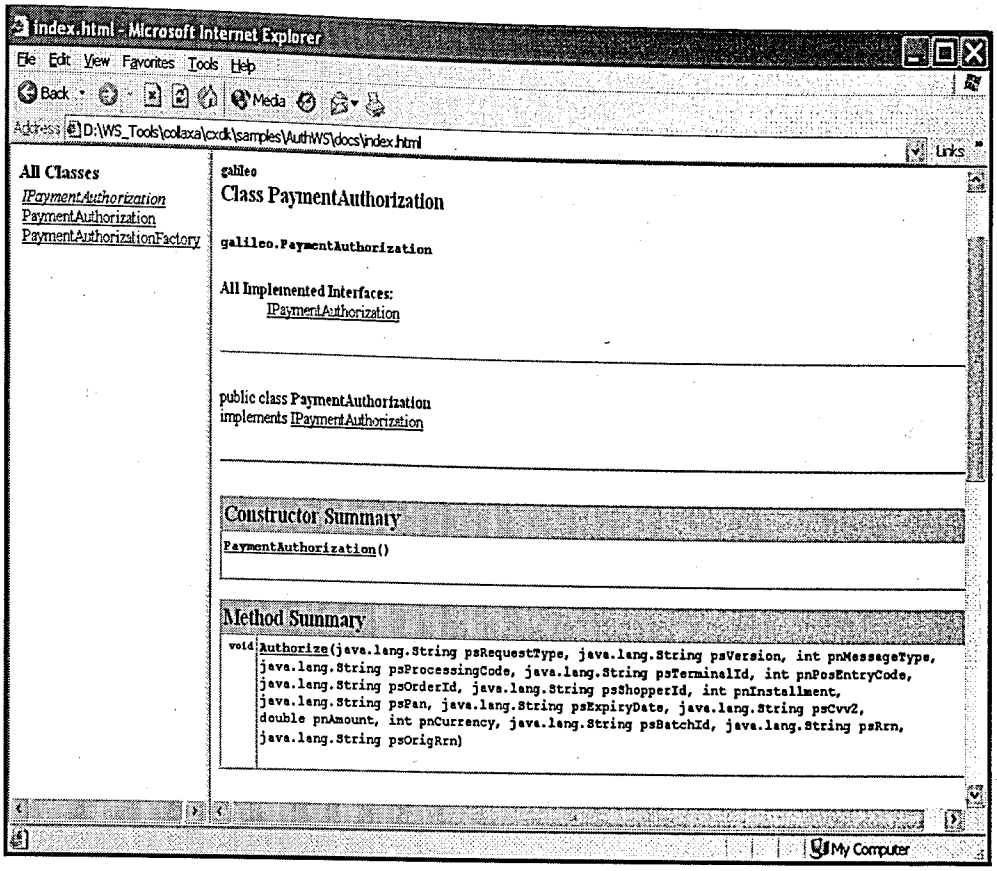

Figure 5.16. Collaxa "wsdlc" utility – Java doc creation

The **"bpelc"** utility converts a BPEL or BPEL Scenario source file and its deployment descriptor into a BPEL Scenario that can be deployed to the server. Besides compiling and packaging the BPEL Scenario

- The WSDL file used to access the BPEL Scenario as a Web service
- A Java Delegate interface for invoking the scenario from JSPs, Servlets and other Java applications are also generated.

Figure 5.17 displays how this utility has been used for deploying the financial use case on the BPEL Orchestration Server.

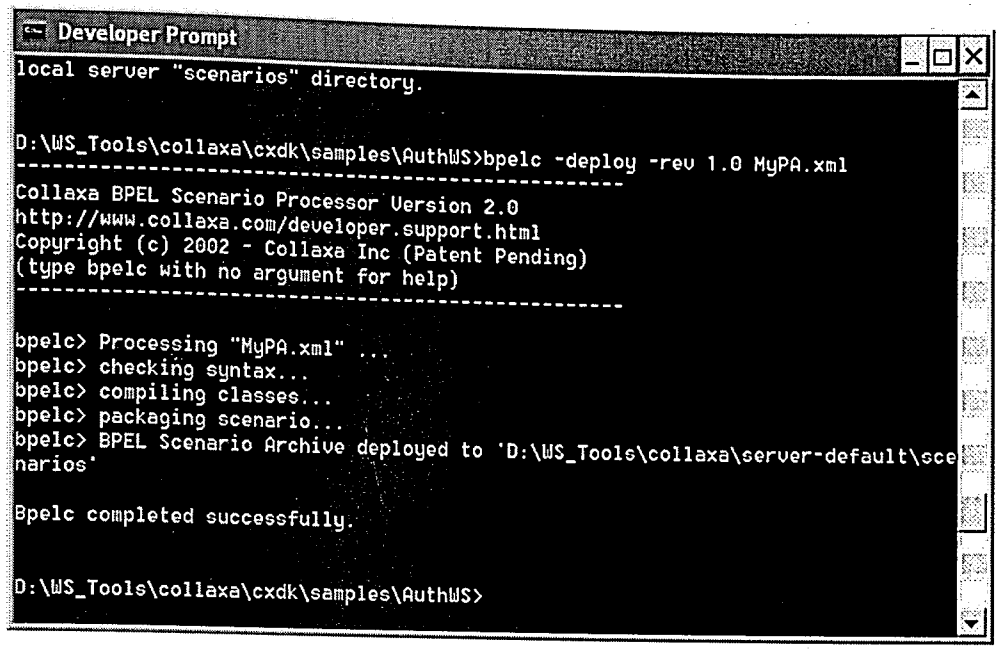

Figure 5.17. Collaxa "bpelc" utility

A service deployed on the BPEL Orchestration Server can be invoked through the ***BPEL Orchestration Console***. This console can be used to initiate and test BPEL Scenarios, debug them, review their audit trail or cancel them. The BPEL Scenario source code can be checked through the source pane in the console, such as in Figure 5.18. Example instances of the "PaymentAuthorization" client, which is actually a BPEL scenario, are shown in Figure 5.19.

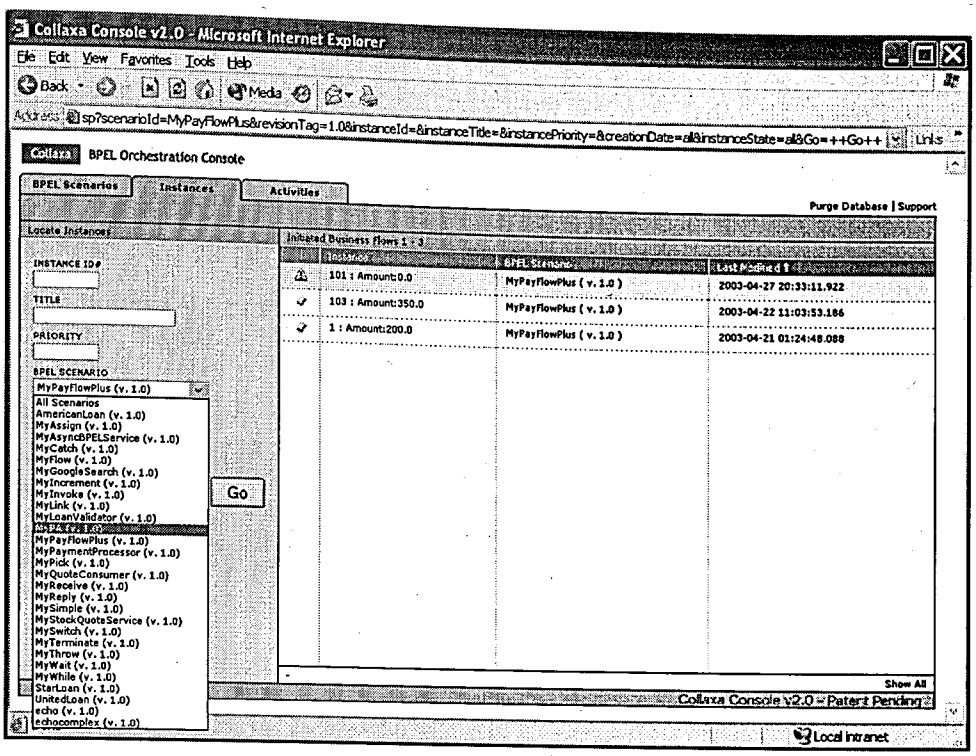

Figure 5.18. BPEL Orchestration Console – BPEL Scenario source pane

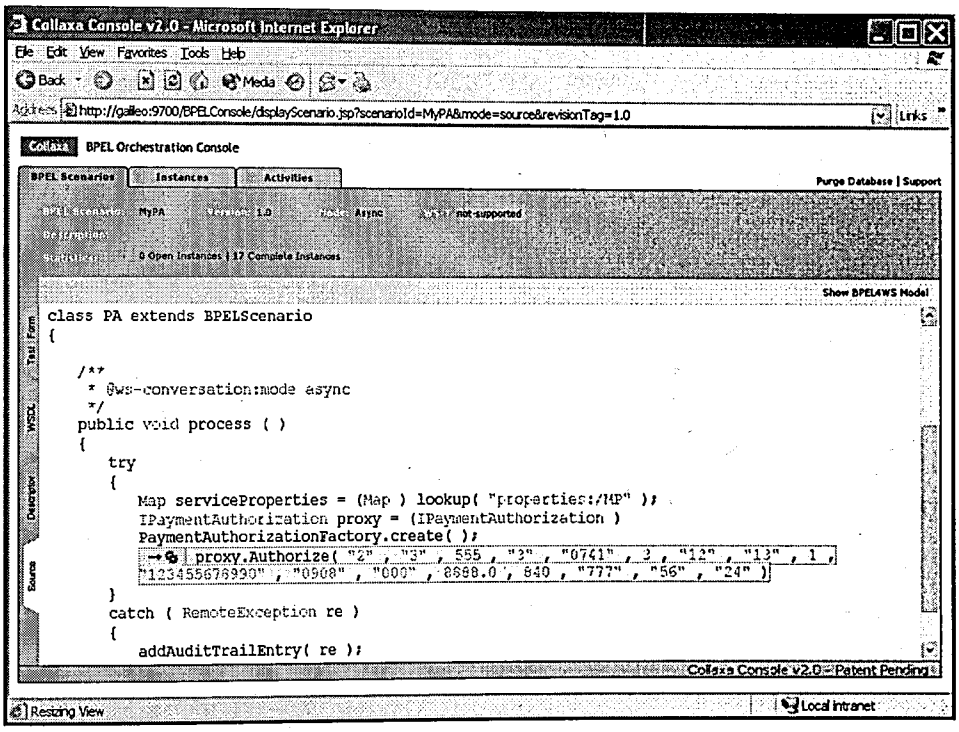

Figure 5.19. Example BPEL Scenario instances

5.5.6. Eclipse BP4WSJ Editor Plug-in

The ***IBM BPWS4J Editor*** is an eclipse plug-in, which allows creating new BPEL4WS processes and manipulating existing BPEL4WS documents. A BPEL4WS process can be visually edited in a "process" editor, which is a structured tree representation of the process or the XML-based BPEL4WS document can be edited in the "source" editor (IBM, 2002b).

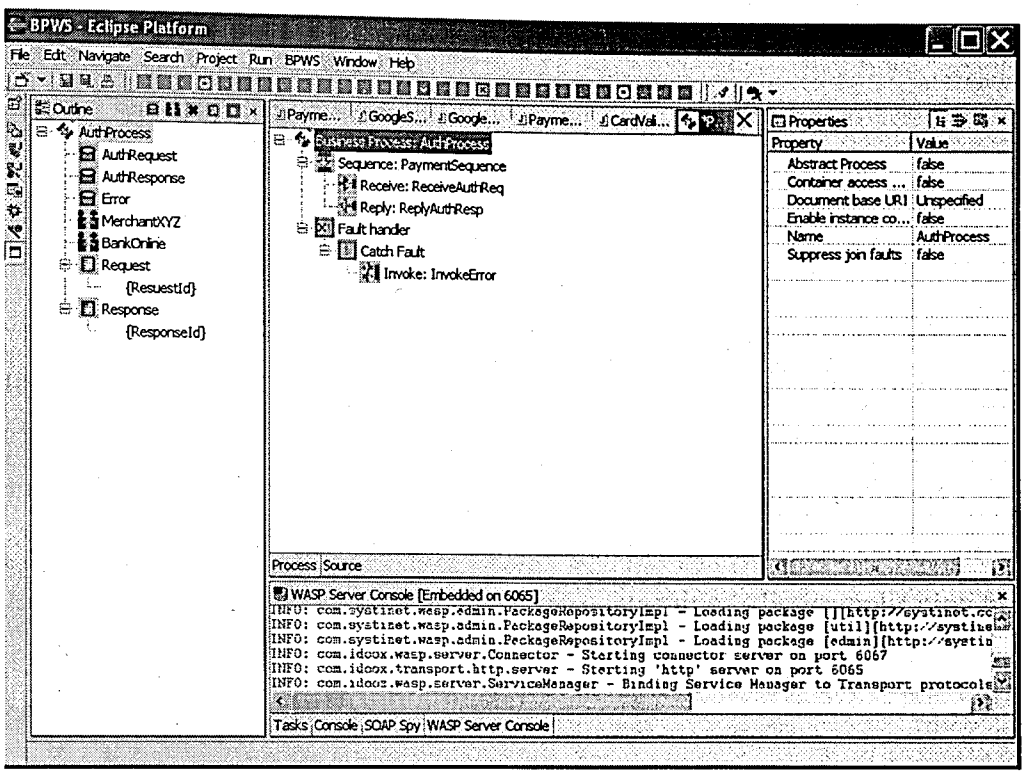

Figure 5.20. BPEL4WS process – visual design

A sample BPEL4WS document has been created for the financial service use case. The top-down design of the BPEL4WS process is displayed in Figure 5.20. The actual BPEL4WS code is listed in Appendix C.

The BPWS4J offers an integrated basic development for business process specific features which will be built into the "PaymentAuthorization" service.

5.6. Final Remarks

Financial institutions offer a variety of services and follow leading trends in IT technology. As financial service processes vary in duration and significance, it is possible to create a diverse range of usage scenarios based on such services and identify the current state of Web service adoption. It is also expected that financial services will lead the way for externally deployed Web Services (Narsu, 2002), so that more sophisticated business needs can also be analyzed.

In general, a payment processing service could be considered as a basic financial operation, which offers a simple request response mechanism, thus resembling initial Web service examples. On the contrary, the sample application exists within a dynamic computing infrastructure and is a real-time process, which has to be completed within seconds (max. 1-2 seconds). In addition, it is a mission-critical application as it directly triggers accounting programs and reports authorization results and response times. To sum up, the authorization application requires full system integration with security and performance considerations. If the initial Web service transformation proves to be successful, service security, performance, composition and management can be listed as future service enhancements. Due to all these reasons, this initially basic application turned into a profound use case to examine the adoption of Web service technologies for financial services on a step-by-step basis.

Although development starts with the Web service transformation of the actual application, if demand for sophisticated internal/external integration solutions with the redesigned service arises, the staged development approach will provide the flexibility to equip the service with new features. Thus it will be possible to build the service within a business process flow and make composite services interact with each other. Moreover, each development and feature enhancement stage gives the option to observe various difficulties of Web service adoption in detail and identify a step-by-step roadmap for the actual implementer.

6. CONCLUSION

6.1. Concluding Remarks

The overall study presents a broad snapshot of the current state of the Web service landscape. Web services are not a revolution in distributed computing; however they are an evolutionary step for integration. Web service concepts already promise to hold a massive potential for distributed computing and application integration, however to reach up to the complete vision of Web services is a long-term issue. Today many Web service examples have API-centric features and demonstrate client-server like communication operations, whereas B2C Web services across enterprises are not commonplace.

The battle for standardized and widely accepted specifications still continues in some Web service areas. Throughout the study many Web service specifications have kept evolving. In the end effect, Web service early adopters had to master more and more Web service acronyms. In addition, we witnessed the convergence of specifications such as XLANG and WSFL to form BPEL4WS and various standardization groups, such as W3C and OASIS, which conduct disparate work for the same Web service area. Currently BPEL4WS gains widespread support. This might be due to the fact that BPEL4WS is a convergence of two promising former standards and now combines their most useful features. Another reason underlying the industry support for BPEL4WS might also be a strong-belief that it will certainly become the de-facto standard as it has been produced by the agreement of companies like Microsoft, IBM, BEA, SAP and Siebel.

6.2. Findings and Recommendations for Web Service Adoption

Due to the in-development areas in Web services, it is possible to foresee that widespread Web service adoption has still a long way to go. Despite this fact, organizations should start to utilize primary Web service standards to become familiar with development concepts and service design methodologies.

The financial use case was primarily selected due to investigation purposes. Mainstream external Web service examples are rare, since many Web service adopters prefer to wrap existing low-value applications into Web services. As core Web service standards are already widely exploited, the Web service transformation for such applications works with no extraordinary effort. However these typical service scenarios and settings are not sufficient to observe Web service adoption and implementation conditions. For that reason, a higher-level service, which provides mission-critical operations and requires customized settings, was needed. The results of the incremental transformation of the use case demonstrated that it is still advisable to wait for external Web service deployment, until specific Web service technologies and tools mature. The use case provides a realistic picture of where we stand today with Web service business practices. Thus, definitely moving to mission-critical applications can only be safely taken into consideration, when higher-level Web service standards have matured and widespread usage of emerging Web services technology takes place.

A starting point for web service adoption could be to make overall analysis of service-oriented and process-centric architecture concepts, before diving into the ocean of Web services standards without any insight. Until a unified approach for specifications is reached, vendor-specific incompatible solutions might arise due to the increasing number of non-standardized specifications in some fields of Web services. While industry-wide standardization efforts continue, companies, who plan to utilize Web service standards, should spend this time on creating custom-roadmaps for their Web services software development life-cycle. Aside from specialized feature requirements or standardization activities, there are only a few real business scenarios and implementations for specifying best-of-breed adoption methods and potential Web service benefits. Even the financial application that appeared to need a straightforward Web service transformation operation, demonstrated unexpected development considerations. Certainly a detailed integration plan would have accelerated the development phase.

6.3. Future Directions

The outcome of the study has already been summarized in detail for several paper submissions. But for demonstration purposes a complete financial process consisting of multiple applications is going to be identified. These applications will be transformed into Web services and process flow will be controlled and applied through Web service orchestration concepts. The "PaymentAuthorization" service outlined in Chapter 5 has already been coded as a BPEL4WS process. As adoption of BPEL4WS also gains industry-wide adoption, probably the rest of the applications might also be converted into BPEL4WS processes. A detailed real business scenario would be a first step towards creating a custom development roadmap for Web service orchestration in the financial services industry.

APPENDIX A: SAMPLE WSDL DOCUMENTS

- **"CardValidator" WSDL document**

  Source: http://webservices.imacination.com/validate/Validate.jws, "Validate.wsdl"

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions
  targetNamespace=http://webservices.imacination.com/validate/Validate.jws
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:apachesoap="http://xml.apache.org/xml-soap"
  xmlns:impl="http://webservices.imacination.com/validate/Validate.jws"
  xmlns:intf="http://webservices.imacination.com/validate/Validate.jws"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <wsdl:message name="validateCardResponse">
      <wsdl:part name="validateCardReturn" type="xsd:boolean"/>
</wsdl:message>
<wsdl:message name="validateNumberRequest">
      <wsdl:part name="ccNumber" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="validateNumberResponse">
      <wsdl:part name="validateNumberReturn" type="xsd:boolean"/>
</wsdl:message>
<wsdl:message name="validateCardRequest">
      <wsdl:part name="ccNumber" type="xsd:string"/>
<wsdl:part name="ccDate" type="xsd:string"/>
</wsdl:message>
<wsdl:portType name="Validate">
      <wsdl:operation name="validateCard" parameterOrder="ccNumber ccDate">
```

```
            <wsdl:input message="impl:validateCardRequest"
            name="validateCardRequest"/>
            <wsdl:output message="impl:validateCardResponse"
            name="validateCardResponse"/>
        </wsdl:operation>
        <wsdl:operation name="validateNumber" parameterOrder="ccNumber">
            <wsdl:input message="impl:validateNumberRequest"
            name="validateNumberRequest" />
            <wsdl:output message="impl:validateNumberResponse"
            name="validateNumberResponse" />
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="ValidateSoapBinding" type="impl:Validate">
        <wsdlsoap:binding style="rpc"
        transport="http://schemas.xmlsoap.org/soap/http"/>
        <wsdl:operation name="validateCard">
            <wsdlsoap:operation soapAction="" />
            <wsdl:input name="validateCardRequest">
                <wsdlsoap:body
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                namespace="http://webservices.imacination.com/validate/Validate.
                jws" use="encoded" />
            </wsdl:input>
            <wsdl:output name="validateCardResponse">
                <wsdlsoap:body
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                namespace="http://webservices.imacination.com/validate/Validate.
                jws" use="encoded" />
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="validateNumber">
            <wsdlsoap:operation soapAction="" />
```

```
            <wsdl:input name="validateNumberRequest">
                <wsdlsoap:body
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                namespace="http://webservices.imacination.com/validate/Validate.
                jws" use="encoded" />
            </wsdl:input>
            <wsdl:output name="validateNumberResponse">
                <wsdlsoap:body
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                namespace="http://webservices.imacination.com/validate/Validate.
                jws" use="encoded" />
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="ValidateService">
        <wsdl:port binding="impl:ValidateSoapBinding" name="Validate">
            <wsdlsoap:address
            location="http://webservices.imacination.com/validate/Validate.jws" />
        </wsdl:port>
    </wsdl:service>
    </wsdl:definitions>
```

- **"CardValidator" Web service client**

```
package tester.CardValidation;

import org.idoox.wasp.Context;
import org.idoox.webservice.client.WebServiceLookup;
import java.util.ArrayList;
import java.util.ListIterator;
import tester.CardValidation.iface.Validate;
public class CardValidatorClient{
```

```
    private static String wsdl =
        "http://webservices.imacination.com/validate/Validate.jws?wsdl";
    private static String service =
        "http://webservices.imacination.com/validate/Validate.jws";
    private static ArrayList nums;
    static{
        nums = new ArrayList();
        nums.add(new CreditCard("5454-5454-5454-5454","03/2003"));
        nums.add(new CreditCard("4444-4444-4444-4448","06/2004"));
        nums.add(new CreditCard("4444-4444-4444-4448","06/2002"));
        nums.add(new CreditCard("5321-7123-5712-3570","11/2003"));
        nums.add(new CreditCard("0000-0000-0000-0000","02/2004"));
    }
    public static void main (String[] args) throws Exception {
        Validate validService;
        CreditCard card;
        WebServiceLookup lookup = (WebServiceLookup)Context.getInstance(
            Context.WEBSERVICE_LOOKUP);
        validService = (Validate)lookup.lookup(wsdl, Validate.class, service);
        ListIterator lit = nums.listIterator();
        while(lit.hasNext()){
            card = (CreditCard)lit.next();
            System.out.print(card.getNumber() + " exp. "+
            card.getExpDate()+" is ");
            if(validService.validateCard(card.getNumber(),
            card.getExpDate()))
                System.out.print("valid");
            else  System.out.print("invalid");
            System.out.println();
        }//while
    }//main
}//class
```

- **Financial Web Service Use Case – WSDL Document**

Tool: Systinet Wasp Developer

Source: PaymentAuthorization.wsdl

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions
name="com.bank.ecommerce.authorization.PaymentAuthorization"
targetNamespace="urn:com.bank.ecommerce.authorization.PaymentAuthorization"
xmlns:tns="urn:com.bank.ecommerce.authorization.PaymentAuthorization"
xmlns:ns0="http://systinet.com/xsd/SchemaTypes/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:map="http://systinet.com/mapping/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
<wsdl:types>
    <xsd:schema targetNamespace=http://systinet.com/xsd/SchemaTypes/
    elementFormDefault="qualified"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://systinet.com/xsd/SchemaTypes/">
        <xsd:element name="p0" type="xsd:string" nillable="true" />
        <xsd:element name="p1" type="xsd:string" nillable="true" />
        <xsd:element name="p2" type="xsd:int" />
        <xsd:element name="p3" type="xsd:string" nillable="true" />
        <xsd:element name="p4" type="xsd:string" nillable="true" />
        <xsd:element name="p5" type="xsd:int" />
        <xsd:element name="p6" type="xsd:string" nillable="true" />
        <xsd:element name="p7" type="xsd:string" nillable="true" />
        <xsd:element name="p8" type="xsd:int" />
        <xsd:element name="p9" type="xsd:string" nillable="true" />
        <xsd:element name="p10" type="xsd:string" nillable="true" />
        <xsd:element name="p11" type="xsd:string" nillable="true" />
```

```
            <xsd:element name="p12" type="xsd:double" />
            <xsd:element name="p13" type="xsd:int" />
            <xsd:element name="p14" type="xsd:string" nillable="true" />
            <xsd:element name="p15" type="xsd:string" nillable="true" />
            <xsd:element name="p16" type="xsd:string" nillable="true" />
        </xsd:schema>
    </wsdl:types>
    <wsdl:message name="PaymentAuthorization_Authorize_Response" />
    <wsdl:message name="PaymentAuthorization_Authorize_1_Request">
        <wsdl:part name="p0" element="ns0:p0" />
        <wsdl:part name="p1" element="ns0:p1" />
        <wsdl:part name="p2" element="ns0:p2" />
        <wsdl:part name="p3" element="ns0:p3" />
        <wsdl:part name="p4" element="ns0:p4" />
        <wsdl:part name="p5" element="ns0:p5" />
        <wsdl:part name="p6" element="ns0:p6" />
        <wsdl:part name="p7" element="ns0:p7" />
        <wsdl:part name="p8" element="ns0:p8" />
        <wsdl:part name="p9" element="ns0:p9" />
        <wsdl:part name="p10" element="ns0:p10" />
        <wsdl:part name="p11" element="ns0:p11" />
        <wsdl:part name="p12" element="ns0:p12" />
        <wsdl:part name="p13" element="ns0:p13" />
        <wsdl:part name="p14" element="ns0:p14" />
        <wsdl:part name="p15" element="ns0:p15" />
        <wsdl:part name="p16" element="ns0:p16" />
    </wsdl:message>
    <wsdl:message
name="PaymentAuthorization_Authorize_java.lang.Exception_Fault">
        <wsdl:part name="idoox-java-mapping.java.lang.Exception"
        type="xsd:string"/>
    </wsdl:message>
```

```
<wsdl:portType name="PaymentAuthorization">
    <wsdl:operation name="Authorize" parameterOrder="p0 p1 p2 p3 p4 p5 p6 p7
    p8 p9 p10 p11 p12 p13 p14 p15 p16">
        <wsdl:input
        message="tns:PaymentAuthorization_Authorize_1_Request"/>
        <wsdl:output
        message="tns:PaymentAuthorization_Authorize_Response" />
        <wsdl:fault name="Authorize_fault1" message=
        "tns:PaymentAuthorization_Authorize_java.lang.Exception_Fault"/>
    </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="PaymentAuthorization" type="tns:PaymentAuthorization">
    <soap:binding transport=http://schemas.xmlsoap.org/soap/http
    style="document" />
    <wsdl:operation name="Authorize">
        <map:java-operation name="Authorize"
        signature="KExqYXZhL2xhbmcvU3RyaW5nO0xqYXZhL2xhbmcvU3
        RyaW5nO0lMamF2YS9sYW5nL1N0cmluZztMamF2YS9sYW5nL1N0c
        mluZztJTGphdmEvbGFuZy9TdHJpbmc7TGphdmEvbGFuZy9TdHJpbm
        c7SUxqYXZhL2xhbmcvU3RyaW5nO0xqYXZhL2xhbmcvU3RyaW5nO
        0xqYXZhL2xhbmcvU3RyaW5nO0RJTGphdmEvbGFuZy9TdHJpbmc7
        TGphdmEvbGFuZy9TdHJpbmc7TGphdmEvbGFuZy9TdHJpbmc7KVY
        =" />
        <soap:operation
        soapAction="urn:com.bank.ecommerce.authorization.PaymentAuthorizat
        ionPaymentAuthorization#Authorize#KExqYXZhL2xhbmcvU3RyaW5n
        O0xqYXZhL2xhbmcvU3RyaW5nO0lMamF2YS9sYW5nL1N0cmluZzt
        MamF2YS9sYW5nL1N0cmluZztJTGphdmEvbGFuZy9TdHJpbmc7TGp
        hdmEvbGFuZy9TdHJpbmc7SUxqYXZhL2xhbmcvU3RyaW5nO0xqYX
        ZhL2xhbmcvU3RyaW5nO0xqYXZhL2xhbmcvU3RyaW5nO0RJTGphd
        mEvbGFuZy9TdHJpbmc7TGphdmEvbGFuZy9TdHJpbmc7TGphdmEvb
        GFuZy9TdHJpbmc7KVY=" style="document" />
```

```
            <wsdl:input>
                <soap:body use="literal" />
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal" />
            </wsdl:output>
            <wsdl:fault name="Authorize_fault1">
            <soap:fault name="Authorize_fault1" use="literal"
                encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
             namespace="urn:com.bank.ecommerce.authorization.Payment
             AuthorizationPaymentAuthorization" />
            </wsdl:fault>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="ccPaymentAuthorization">
        <wsdl:port name="PaymentAuthorization"
        binding="tns:PaymentAuthorization">
            <soap:address location="http://localhost:6060/PaymentAuthorization/" />
        </wsdl:port>
    </wsdl:service>
    </wsdl:definitions>
```

- **Financial Web Service Use Case – WSDL Document**

Tool: CapeClear Studio/ WSDL Editor
Source: wsPaymentApps.wsdl

```
<?xml version="1.0" encoding="UTF-8" ?>
<definitions name="wsPaymentApps"
targetNamespace="http://galileo:8000/wsPaymentApps.wsdl"
xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/"
```

```
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://galileo:8000/wsPaymentApps.wsdl"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsd1="http://galileo:8000/wsPaymentApps.xsd">
<message name="AuthorizeResponse" />
<message name="Authorize">
    <part name="psRequestType" type="xsd:string" />
    <part name="psVersion" type="xsd:string" />
    <part name="pnMessageType" type="xsd:int" />
    <part name="psProcessingCode" type="xsd:string" />
    <part name="psTerminalId" type="xsd:string" />
    <part name="pnPosEntryCode" type="xsd:int" />
    <part name="psOrderId" type="xsd:string" />
    <part name="psShopperId" type="xsd:string" />
    <part name="pnInstallment" type="xsd:int" />
    <part name="psPan" type="xsd:string" />
    <part name="psExpiryDate" type="xsd:string" />
    <part name="psCvv2" type="xsd:string" />
    <part name="pnAmount" type="xsd:double" />
    <part name="pnCurrency" type="xsd:int" />
    <part name="psBatchId" type="xsd:string" />
    <part name="psRrn" type="xsd:string" />
    <part name="psOrigRrn" type="xsd:string" />
</message>
<portType name="PaymentAuthorization">
    <operation name="Authorize">
        <input message="tns:Authorize" />
        <output message="tns:AuthorizeResponse" />
    </operation>
</portType>
<binding name="PaymentAuthorizationBinding" type="tns:PaymentAuthorization">
```

```
        <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
        <operation name="Authorize">
            <soap:operation soapAction="" />
            <input>
                <soap:body
                namespace="capeconnect:wsPaymentApps:PaymentAuthorization"
                use="literal" />
            </input>
            <output>
                <soap:body
                namespace="capeconnect:wsPaymentApps:PaymentAuthorization"
                use="literal" />
            </output>
        </operation>
</binding>
<service name="wsPaymentApps">
        <port binding="tns:PaymentAuthorizationBinding"
        name="PaymentAuthorization">
            <soap:address location="http://galileo:8000/ccx/wsPaymentApps" />
        </port>
</service>
</definitions>
```

APPENDIX B: FINANCIAL USE CASE WEB SERVICE CLIENTS

- **Systinet Wasp Developer – Payment Authorization Web Service Client**

Tool: Systinet Wasp Developer – Eclipse plugin
Source: PaymentAuthorizationClient.java

```
package com.bank.ecommerce.authorization;

import org.idoox.webservice.client.WebServiceLookup;
import org.idoox.wasp.Context;

import com.bank.ecommerce.authorization.iface.PaymentAuthorization;

public class PaymentAuthorizationClient extends Object {
    /** Constructor */
    public PaymentAuthorizationClient() {}
    /** @param args the command line arguments */
    public static void main (String args[]) throws Exception {
    com.bank.ecommerce.authorization.iface.PaymentAuthorization service;
        String serviceURI = "http://localhost:6060/PaymentAuthorization/";
        String wsdlURI = "http://localhost:6060/PaymentAuthorization/";
        // init the lookup
        WebServiceLookup lookup = (WebServiceLookup)
        Context.getInstance(Context.WEBSERVICE_LOOKUP);
        // get the instance of the Web Service interface from the lookup
        service =
        (com.bank.ecommerce.authorization.iface.PaymentAuthorization)
        lookup.lookup(wsdlURI,
        com.bank.ecommerce.authorization.iface.PaymentAuthorization.class,
        serviceURI);
```

```
            //call the methods on your Web Service interface
            service.Authorize("authreq","001.0",100,"553000", "00030675", 812,
            "666666", "987654",3,"5499988000820301","0310",
            "899",1200000,792,"123456789","tester1234567","");
        }
}//PaymentAuthorization - client
```

- **CapeClear Studio – Payment Authorization Web Service Client**

Tool: CapeClear Studio – Payment Authorization Web service client
Source: wsPaymentAppsClient.java

```
package wspaymentapps;

public class WsPaymentAppsClient {
    /*Construct a new WsPaymentAppsClient */
    public WsPaymentAppsClient ( ) {  }

    /*Run standalone */
    public static void main( java.lang.String[] args ) {
    PaymentAuthorization proxy = null;
    try {
        System.out.println("Setting up call to method : authorize");
        proxy = PaymentAuthorizationFactory.create();
        proxy.authorize("", "", (int)0, "", "", (int)0, "", "", (int)0, "", "", "",
        (double)0.0, (int)0, "", "", "");
    }
    catch( java.lang.Throwable exp ) {
        System.out.println("Exception occurred : authorize");
        System.out.println( exp.getMessage() );
        exp.printStackTrace(System.out); }
    finally {
```

```
            ((com.capeclear.capeconnect.soap.proxy.dynamic.DynSoapProxy)proxy)
            .dispose();
            System.out.println("Finished Invoking call to method : authorize");
        }//finally
        }//main
}//class
```

- **Collaxa BPEL Orchestration Server – Payment Authorization Web Service Client**

Tool: Collaxa BPEL Orchestration Server/ BPEL Scenario
Source: PaymentAuth.jbpel

```
/*
*IPaymentAuthorization is the 2-way proxy for the
*{http://galileo:8000/PaymentAuthorization.wsdl}
*PaymentAuthorization web service.
*The code is used to test the Web Service usage within a BPEL Scenario.
*/
package galileo;
import java.rmi.RemoteException;
import java.util.Map;
import com.collaxa.bpel.BPELScenario;
import galileo.IPaymentAuthorization;
import galileo.PaymentAuthorizationFactory;

public class PaymentAuth extends BPELScenario {
    /*
    *    @ws-conversation:mode async
    */

    public void process( ){
```

```
try{
        // Get properties for the service plus endpoint location. Properties have
        been externalized in the deployment descriptor of the BPEL Scenario
        Map serviceProperties = (Map) lookup( "properties:/MP" );

        // Create an instance of the 2-way Web Service proxy.
        IPaymentAuthorization proxy = (IPaymentAuthorization)
        PaymentAuthorizationFactory.create( serviceProperties );

        // Invoke operation on the proxy.
        //This code will create the appropriate SOAP message and send it to the
        //remote Web service
        proxy.Authorize("authreq","001.0",100,"553000", "00030675", 812,
        "666666", "987654",3,"5499988000820301","0310",
        "899",1200000,792,"123456789","tester1234567","");
    }
    catch ( RemoteException re ){
        // messages from the remote XML Web Service. Log Exception
       // into audit trail of this instance
      addAuditTrailEntry( re );
    }//catch
    }//method
}//class
```

APPENDIX C: FINANCIAL USE CASE BPEL4WS DOCUMENT

- **Business Process Execution Language for Web Services Java Run Time (BPWS4J) Example**

Tool: Eclipse – BPEL4WS Editor
Source: PaymentAuth.bpel

```
<process xmlns="http://schemas.xmlsoap.org/ws/2002/07/business-process/"
name="AuthProcess"
targetNamespace="http://tempuri.org/"
xmlns:tns="http://tempuri.org/"
suppressJoinFailure="no"
containerAccessSerializable="no"
enableInstanceCompensation="no"
abstractProcess="no">
<partners>
    <partner name="MerchantXYZ" myRole="Merchant" partnerRole="Bank"/>
    <partner name="BankOnline" myRole="Bank" partnerRole="Merchant"/>
</partners>
<containers>
<container name="AuthRequest"/>
    <container name="AuthResponse"/>
    <container name="Error"/>
</containers>
<correlationSets>
    <correlationSet xmlns:ns1="ResuestId" name="Request" properties="ns1:"/>
    <correlationSet xmlns:ns2="ResponseId" name="Response"
    properties="ns2:"/>
</correlationSets>
<faultHandlers>
```

```
        <catch xmlns:ns3=http://schemas.xmlsoap.org/ws/2002/07/business-process/
        faultName="ns3:AuthorizationError" faultContainer="AuthResponse">
        <invoke name="InvokeError"
                partner="MerchantXYZ" operation="sendError"
                inputContainer="Error" outputContainer="AuthResponse">
        </invoke>
</catch>
</faultHandlers>

<sequence name="PaymentSequence">
        <receive name="ReceiveAuthReq"
        partner="MerchantXYZ" operation="request"
        container="AuthRequest" createInstance="yes">
        </receive>
        <reply name="ReplyAuthResp"
                partner="MerchantXYZ" operation="reply"
container="AuthResponse">
        </reply>
</sequence>
</process>
```

REFERENCES

Ajay, M. R., 2002, *Web Services: A Detailed Overview*, EvolveWare, Inc, http://www.evolveware.com

Aldrich, S. E., 2002, *Systinet Web-Services Deployment: WASP Server for Java Web Application and Services Platform for Development, Assembly, Discovery, and Deployment*, Patricia Seybold Group - Strategic Research Service

Anbazhagan, M. and A. Nagarajan, 2002, *Understanding Quality of Service for Web Services*, IBM developerWorks Web Services Articles.

Assaf, A., S. Askary, S. Fordin, W. Jekeli, K. Kawaguchi, D. Orchard, S. Pogliani, K. Riemer, S. Struble, P. Takacsi-Nagy and S. Zimek, 2002, *Web Service Choreography Interface 1.0*, Intalio, Sun Microsystems, SAP, BEA Systems

Bloor Research, July 2002, *Web Services Gotchas, How Enterprises Can Build Secure, Reliable, Performance-optimized Web Services Solutions While Waiting for the Standards to Mature*, Bloor Research, North America.

Brittenham, P. and D. Davis, 2003, *Web Services Development Concepts 2.0*, IBM Software Group.

CapeClear, 2002, *Cape Clear 4 Technical Overview*, http://www.capeclear.com/products/whitepapers/CapeClear4TechnicalOverview.pdf

Cerami, E., 2002, *Web Service Essentials*, O'Reilly.

Collaxa, 2002, *Web Service Orchestration Server 2.0: An Introduction*, http://www.collaxa.com/pdf/tech_wp.pdf.pxml

Collaxa, 2002, *Collaxa Technology*, http://www.collaxa.com/product.technical.html

Collaxa, 2003, BPEL *Developer's Guide 2.0 Beta-8,* http://www.collaxa.com/developer.download.dev20latest.html.pxml

Curbera, F., Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte and S. Weerawarana, 2002, *Business Process Execution Language for Web Services. Version 1.0*, BEA, IBM, Microsoft.

Dubray, J.J., 2003, *Business Process Definition Languages*, http://www.ebpml.org

ebxml, 2002, ebXML.org website, UN/CEFACT, OASIS, http://www.ebxml.org

Eclipse, 2003, *Eclipse JAVA IDE*, http://www.eclipse.org

Gisolfi, D., 2001, *Web Services Architect, Part 1: An Introduction to Dynamic E-Business*, IBM developerWorks Web Services Articles.

Gisolfi, D., 2001, *Web Services Architect: Is Web Services the Reincarnation of CORBA?*, IBM developerWorks Web Services Articles.

Gopalakrishnan, U. and R. K. Ravi, 2003, *Securing Web services*, IBM developerWorks Web Services Articles.

Graham, S., 2002, *Building Web Services with Java: Making Sense of XML, SOAP, WSDL and UDDI*, Sams Publishing.

Gunzer, H., 2002, *Introduction to Web Services*, Borland, http://www.borland.com/webservices/white_papers/

Goethals, F., 2002, *New Directions in Application Integration: Web Services,* Katholieke Universiteit Leuven, Belgium.

Haberl, S., 2003, *Business Process Description Languages*, http:// www.cis.unisa.edu.au/~cissh /research/webflow/index.html

IBM, 2002, *Security in a Web Services World: A Proposed Architecture and Roadmap*, Security whitepaper from IBM Corporation and Microsoft Corporation

IBM, 2002, *BPWS4J Editor*, IBM alphaWorks.

integra, 2002, *A Definition of Web Services*, integra – Genuity, http://www.integra.net.uk/ services

Kreger, H. 2001, *Web Services Conceptual Architecture*, IBM Software Group

Leymann, F., 2001, *Web Services Flow Language 1.0*, IBM Software Group, http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

Leymann, F. and D. Roller, 2002, *Business Processes in a Web Services World*, IBM developerWorks Web Services Articles.

Leymann, F., D. Roller and M.T. Schmidt, 2002, "Web Services and Business Process Management", *IBM Systems Journal*, Vol . 41, No 2

Narsu, U. and P. Murphy, 2002, *Web Services Adoption Outlook Improves*, Giga Information Group, http://www.gigaweb.com

Newcomer, E., 2002, *Understanding Web services*, Addison-Wesley.

OASIS, 2003, *Web Services Business Process Execution Language (WSBPEL)-Call for Participation*, http://lists.oasis-open.org/archives/tcannounce/200304, OASIS WS-BPEL Technical Committee

Ogbuji, U., 2002, *The Past, Present and Future of Web Services*, Webservices.org articles

O'Riordan, D., 2002, *Business Process Standards For Web Services*, Tect, http:// www.webservicesarchitect.com.

Peltz, C., 2003, *Web Services Orchestration*, Hewlett Packard Corporation, http:// devresource.hp.com/drc/technical_white_papers/WSOrch/WSOrchestration.pdf

Schmale, T., 2002, "Web Services between Demand and Reality", *Net.ObjectDays*, inubit AG, http://www.inubit.de

Sleeper, B. and B. Robins, 2002, *The Laws of Evolution: A Pragmatic Analysis of the Emerging Web Services Market*, Stencil Group Analysis Memo.

Snell, J., 2001, *Introducing the Web Services Flow Language*, IBM developerWorks Web Services Articles.

Snell, J., 2002, *Automating Business Processes and Transactions in Web Services*, IBM developerWorks Web Services Articles.

Systinet, 2002, *Wasp Developer - Eclipse plugin*, http://www.systinet.com

Thatte, S., 2001, *XLANG Web Services for Business Process Design*, Microsoft Corporation, http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm

Uddi.org, 2000, *UDDI Executive White Paper*, Uddi.org

Uddi.org, 2001, *UDDI Programmer's API Specification*, Uddi.org

van der Aalst, W.M.P., M. Dumas, A.H.M. ter Hofstede and P. Wohed, 2003, *Pattern Based Analysis of BPML (and WSCI)*, FIT Technical Report FIT-TR-2002-05, Eindhoven University of Technology, Queensland University of Technology, The Royal Institute of Technology.

van der Aalst, W.M.P., 2003, "Don't go with the flow:Web services composition standards exposed", *IEEE Intelligent Systems*, January/February Issue 2003.

Virdell, M., 2003, *Business processes and workflow in the Web services world*, IBM developerWorks Web Services Articles.

W3C, 2000, *Extensible Markup Language 1.0*, W3C Recommendation, http:// www.w3.org/TR/2000/REC-xml-20001006

W3C, 2001, *Web Services Description Language 1.1*, W3C Working Draft, http:// www.w3org/TR/wsdl

W3C, 2001, *XML Schema Part 2: Datatypes,* W3C Recommendation, http://www.w3.org/ TR/xmlschema-2/

W3C, 2002, *SOAP Version 1.2*, W3C Working Draft, http://www.w3.org/TR/soap12-part0

W3C, 2002, *Web Services Description Language 1.2.*, W3C Working Draft, http://www. w3.org/TR/2003/WD-wsdl12-20030124/

W3C, 2002, *Web Services Architecture Requirements*, W3C Working Draft, http:// www.w3.org/TR/2002/WD-wsa-reqs-20021011

W3C, 2003, *Web Services Glossary*, W3C Editor's Draft, http://www.w3.org/TR/ws-gloss/

Weerawarana, S. and F. P. Curbera, 2002, *Business Processes: Understanding BPEL4WS/ Concepts in Business Processes*, IBM developerWorks Web services articles

WfMC, 1999, *Workflow Management Coalition: Terminology & Glossary*, Workflow Management Coalition, Document Number WFMC-TC-1011.

Wilkes, L., 2002, *Web Services – Right Here, Right Now*, CBDi Forum, http:// www.cbdiforum.com

Wohed, P., W.M.P. van der Aalst, M. Dumas and A.H.M. ter Hofstede, 2003, *Pattern Based Analysis of BPEL4WS*, Technical Report FIT-TR-2002-04, The Royal Institute of Technology, Eindhoven University of Technology, Queensland University of Technology.

Zakheim, B., 2002, *Web Services Integration: The Next Generation of Integration Platforms*, IONA Technologies.

REFERENCES NOT CITED

Patlak, Ç., A. B. Bener and H. Bingöl, 2003, "Web Service Standards and Real Business Scenario Challenges", *29th Euromicro Conference Component-based Software Engineering Track, Web Service Engineering Session Short Paper*, 1-5 September 2003, Antalya.

Spek, W. A., 2002, *Designing Web Services in a Business Context: A Transformation from Conceptual Business Processes to Web Services*, M.S. Thesis, Tilburg University and Infolab CentER Applied Research, The Netherlands.

SUN, 2002, *Web Services Choreography Interface Editor*, http://www.sun.com/software/xml/developers/wsci/download/

Tosic, V., B.Pagurek, B. Esfandiari and K. Patel, 2001, "On the Management of Compositions of Web Services", Workshop on Object-Oriented Web Services OOWS – (OOPSLA), Tampa, USA, October 2001.

Tosic, V., K. Patel and B.Pagurek, 2002, "Web Service Offerings Language", *Proceedings of the Workshop on Web Services, e-Business, and the Semantic Web at CaiSE*, Toronto, Canada, May 2002.

W3C-Chor, 2003, *W3C-Web Service Choreography Group*, http://www.w3.org/2002/ws/chor