UNIVERSITY OF OSLO
Department of Informatics

Web Services / Distributed Systems

Can Web Services Be Used as Foundation for Distributed Systems?

Solvor Jenny B. Skaaden

June 2004

Abstract

Distributed systems have been a part of computer science for decades. They are systems where one or more computers or devices communicate with other computers or devices. Such a system ideally has a dynamic life where other systems may join or quit at any time without the whole system failing. The communication between the components happens only by passing messages. The technology of web services on the other hand, is a relatively new development. It is based upon the principles of distributed systems. A web service is a set of functions that are published to a network for use by other programs. Many people regard web services as a technology only for publishing software services on the Internet via browsers, while others regard them as the "new big thing" in distributed computing that is working as general purpose architectures. This thesis will analyse both technologies to see if web services can be used as a foundation for distributed systems.

Preface

The thesis

In this thesis I will analyse distributed systems and web services to see if web services can be used as a foundation for distributed systems. I will look at the theoretical and practical aspects of both distributed systems and web services.

Problem description and organising

The given problem description was *web services / distributed systems. Can web services be used as foundation for distributed systems?* The following topics should also be included in the thesis: *Overview of distributed systems*, *overview of web services*, *attempt to develop web services*, and *evaluation*. I have used the given problem description and contents as a basis for dividing the topic into chapters.

Distributed systems

Distributed systems are described in chapter 1. They are defined to be system where different components in a network, communicate with each other and coordinate their actions only by passing messages. A component may be a program execution on a computer or a device such as a computer or a printer. It is a rather simple definition, but it covers the entire range of systems that can be called distributed systems. In chapter 1 both the general characteristics and some of the specific characteristics of distributed systems are explored.

Web services

Web services are described in chapter 2. It is a relatively new development. Web services are based upon the principles of distributed systems and are defined to be sets of functions that are published to a network for use by other programs. Many people regard web services as a technology only for publishing software services on the Internet via browsers, while others regard them as the "new big thing" in distributed computing that is working as general purpose architectures. In chapter 2 both the general characteristics and some of the specific characteristics of web services are explored.

Web services development

Chapter 3 will describe my attempts to develop two distributed systems and their web services counterpart. The distributed applications will be developed using Java RMI which I have previous practical knowledge of. The web service solutions will be my first attempt to develop web service applications. The applications and their complete source code are on the enclosed CD. In chapter 4 I will look at the development solutions in chapter 3 and try to evaluate them.

Evaluation

In chapter 5 I will evaluate the previous chapters. I will look at both the theoretical, chapters 1 and 2, and the practical, chapters 3 and 4. The intention is to try to find an answer to the given problem description *can web services be used as foundation for distributed systems*. I will also evaluate the process of writing this thesis. Here I will try to sum up the process as well as try to comment on my own work.

Acknowledgements

Many people have been of great help and support during the work on this thesis. A great deal of thanks goes to my supervisor Birger Møller-Pedersen for guiding and helping me completing the thesis. I would like to thank the Department of Informatics at the University of Oslo where this thesis is undertaken. Many thanks also to Geir Martin Haarberg for his help with the linguistics.

Contents

List of Figures

List of Tables

Chapter 1

Distributed Systems

1.1 Introduction

In this chapter I will describe distributed systems. I have used the book by Coulouris et al [C^+01] as a basis for the chapter, and it is used as a reference where nothing else is indicated. I will recommend this book to those who are interested in learning more about distributed systems.

A distributed system is defined to be a system consisting of different components in a network. These components communicate with each other and coordinate their actions only by passing messages. A component may be a program execution on a computer, a computer, a printer or some other device. It is a rather simple definition, but it covers the entire range of systems that can be called distributed systems.

The design and construction of distributed systems are highly motivated by the desire to share resources. Resources are the "things" that one can share in a networked computer system. It may be hardware components, such as printers, or software components, such as files.

The system appears to the user as one centralized system although it may be spread out through multiple and independent systems that are working together. This makes management and administration a lot more complex compared to centralized systems. It is also more difficult to locate problems or failures since these may be located in any part of the system, which requires relatively advanced error detection and handling. However the advantages will in many cases make up for the disadvantages. The distributed systems are much more flexible than the centralized systems. It is easy for devices to connect to the system. The system can easily be scaled up by adding new components, and upgrades can be done incrementally. Since the distributed systems are based on several computers, the systems can tolerate failures at multiple locations. When an error has occurred, they will still continue to function, although usually in a reduced manner. [Mum01, Ch 1.2]

There are some key characteristics and consequences of distributed systems which give a more detailed description than the limited definition above. I will try both to highlight the general characteristics and consequences of all distributed systems as well as describe in more detail some examples of distributed systems.

1.2 General Characteristics

The distributed systems work outside the normal boundaries of a single system. Although it appears as if it is running only on a single processor, it can be spread out across a network consisting of several machines and processors. The computers work together as one system even if they are situated remotely from each other. This way the resources are more available and reliable, and can therefore be used better and more efficiently. If one of the computer crashes, the others continue working. By storing data in multiple locations, the reliability is improved. [Won03, Lec. 1]

Resource sharing Resources can be shared so anyone in the system can make use of accessible data, software and hardware anywhere in the system. The various systems do not need all the features of a centralized system; they can receive what they lack from the other systems in the network. This way one can use relatively simple computers, which is both cheaper and easier to build than the more complex ones. The sharing of hardware resources, such as printers and hard disks, reduces cost greatly and makes it possible to share data resources and other software resources, such as files, shared databases and web pages. These resources are all implemented on the shared disks and processors.

The resource may be shared in only small closed groups or on the Internet throughout the world. Various resource managers, called *service*, keep control on access and synchronization. These managers are in reality program modules which manage and control resources of certain types. The managers use models for describing how the resources are available, how they can be used, and how the service provider and user can interact. The model can either be client-server based or object based. In a client-server based model, the servers provide certain services (procedures or subroutines) and the clients send queries to the servers asking for these services. In an object based model the resources are modelled as objects, and the operations of the entities in the objects are accessed through interfaces. [C$^+$01, Ch 1.3] [LE03, Lec. 1]

Heterogeneity There may be variations and differences between components in a distributed system. These may cause challenges which ought to be defeated in order for the system to work properly. The systems may

handle these challenges in a variety of ways depending on the design and implementation of the system. The variations and differences may apply to the following:
Network: The network protocol may be implemented over various networks.
Computer hardware: There may be differences in data representation on different processors.
Operating system: The API to the Internet protocol may vary.
Programming languages: There may be differences in data structures and characters.
Implementation by different developers: There must be ways for the programs to communicate with each other. Programs made by different developers must use a common standard to make this possible.
[C$^+$01, Ch 1.4] [LE03, Lec. 1]

Openness A computer system's ability to expand and re-implement is determined by the systems openness. In a distributed system this is usually measured by how well new resources can be added and made available for use by everyone in the network. Since the computers are working together as one system, it is easier to add more resources and power by adding it step by step to the various computers. This requires that new components must be able to integrate with existing components, which requires a uniform inter process mechanism. This is usually done by well defined and published interfaces. However this is just one step in adding and extending resources in a distributed system. There may be a high complexity in a system that involves a variety of components that are designed, implemented and managed by different people. This represents a challenge for the designers. It may also be difficult to keep a clean program structure due to the integration. [C$^+$01, Ch 1.4] [LE03, Lec. 1] [WH03, Lec. 1]

Scalability Distributed systems operate on different scales ranging from small intranets with just two computers to the whole Internet. The systems scalability is the systems ability to handle increase in the amount of resources and number of users. And it is described as *scalable* if it will remain effective, both in performance and in resource use, after a significant increase of resources and users.
There is some challenges in designing and implementing a scalable distributed system:

Controlling costs of resources: It should be possible to expand the systems resources without increasing the costs dramatically. For a system to be resource-scalable, the amount of physical resources should be proportional to the number of users in the system. To use Coulouris et al's example: "if a single file server can support 20 users, then two such servers should be able to support 40 users." [C^{+}01, p. 20] This may seem obvious, but it may not be so easy to accomplish in reality.

Controlling performance loss: The increase in size in any distributed system will result in some performance loss. Performance is measured by the time it takes to access a resource. In hierarchical structured system, this should not exceed O(log n), where n is the data set size, if the system is to be performance-scalable.

Preventing the systems software resources from running out: When a distributed system is designed, considerations should be made concerning the dimensions of the resources in the future. This is to ensure that the system can handle future requirements. For example the supply of available Internet addresses will probably run out in the near future, this is because it was decided to use 32 bits for this purpose. The address space will now be expanded to 128 bits to mend the problem. One should be careful with overcompensating for future expansion, as this may cause other problems such as demands for more storage.

Avoiding bottlenecks: There may be a challenge in maintaining scalability in a system by avoiding bottlenecks. For example may a single server or directory for all users in a system be a serious bottleneck. To avoid this, decentralized algorithms is needed. Partitioning, caching and replication are good examples of relatively efficient remedies to avoid bottlenecks.
[C^{+}01, Ch 1.4] [LE03, Lec. 1] [WH03, Lec. 1]

Fault tolerance Hardware, software and networks may fail. A failure may produce incorrect results, no results at all or prevent information from reaching its desired destination. Classes of failures are listed in table 1.1. It is not desirable that a fault which occurred in one part of the system will affect the other parts. In a distributed system faults occur partially, some components may fail while others continue working. Because of this the handling of failures will happen partially and is more difficult than in centralized systems. There are various techniques for dealing with faults. Here are some of them:

Detecting failures: There are ways of detecting some failures while other failures are more or less impossible to detect. One of the ways to detect failures such as corrupted data is to use checksums in messages and files.

Masking failures: Some detected failures can be masked or made less severe, although the various techniques may not work in worst case scenarios. Two examples of hiding failures are to retransmit messages that fail to arrive and to write file data to two disks instead of just one. A way of making a failure less severe is to simply drop a corrupted message; it may be retransmitted if the sender has not received a confirm message.

Tolerating failures: Some faults can just be tolerated without trying to mask them in any way. For example, a web browser which cannot obtain contact with a web server informs the users about this instead trying to make contact while the user waits.

Recovery: Some software is designed to be able to recover stored data in case of a failure. For example in a database one can "roll back" to the previous committed state without any new changes taking effect, or one can store the changes by committing.

Redundancy: A service should be replicated independent of faults. To use one of Coulouris et al's examples: "There should always be at least two different routes between any two routers in the Internet." [C^+01, p. 22] [C^+01, Ch 1.4] [LE03, Lec. 1]

Class of failure	*Affects*	*Description*
Fail-stop	Process	Process halts and remains halted. Other processes may detect this state.
Crash	Process	Process halts and remains halted. Other processes may detect this state.
Omission	Channel	A message inserted in an outgoing message buffer never arrives at the other end's incoming message buffer.
Send-omission	Process	A process completes a *send*, but the message is not put in the outgoing buffer.
Receive-omission	Process	A message is put in a process's incoming message buffer, but that process does not receive it.
Arbitrary (Byzantine)	Process or channel	Process/channel exhibits arbitrary behaviour: it may send/transmit arbitrary messages at arbitrary times, commit omissions; a process may stop or take an incorrect step.

Table 1.1: Classes of failures. [C^+01, Ch. 2]

Concurrency In a distributed system various components can be executed simultaneously and these components may try to access and update the same resources at the same time. Hence access to resources must be managed to maintain integrity of the system. If not simultaneous updates can occur, which can result in lost updates and inconsistent analysis. One way to do this is to process only one client request at the time, but this reduces throughput. Thus most services and applications allow multiple client requests to be processed simultaneously. To maintain integrity in a distributed system concurrency control is required to synchronize concurrent access to the same resources. [C$^+$01, Ch 1.4] [LE03, Lec. 1]

Transactions A sequence of operations that transfers data from one consistent state to another is called a transaction. They are used in both clients and servers. Several transactions can occur at the same time and some of them may try to access the same resources, see the paragraph above about concurrency. The goal of a transaction is to control access to shared resources and that either the transaction is completed or that nothing has happened to the data. This is ensured by the transactions ACID characteristics. They are:
A - Atomicity - Either all the actions of the transaction are applied, or none at all. This also means that the transaction is to take place without interference.
C - Consistency - The system is at a consistent state when the transaction has completed.
I - Isolation - The temporary results of a transaction are not visible to other transactions.
D - Durability - All the effects of the transaction are saved in permanent storage when it is completed.
To guarantee all of these characteristics, state logs are kept and locks, timestamps, and optimistic concurrency control are used during the transaction. When the system uses locks, there is a chance of deadlock and this must be prevented. See table 1.2 for a list of the various types of locks. When using timestamps, the server records the time of each reading and writing request. Then it uses the time to determine whether it should be done immediately, be delayed, or be rejected. The optimistic concurrency control assumes that conflicts rarely happen, and does not use locks. If a conflict has occurred, the server aborts and the client usually restarts. [C$^+$01, Ch 12-13] [LE03, Lec. 8] [WH03, Lec. 18] [Won03, Lec. 11]

Operations of different transactions		*Conflict*	*Reason*
read	read	No	Because the effect of a pair of *read* operations does not depend on the order in which they are executed.
read	write	Yes	Because the effect of a *read* and a *write* operation depends on the order of their execution.
write	write	Yes	Because the effect of a pair of *write* operations depends on the order of their execution.

Table 1.2: Types of locks and conflicts. [C$^+$01, Ch. 12]

Time In a computer system, time is important. In a distributed system this is no less important, but it is somewhat problematic. Algorithms and applications depend on time to coordinate events and timestamps are used to serialize transactions, validate authentication certificates, keep consistency of distributed data, and to remove duplicates. Every computer has its own physical clock, but it is not possible to synchronize clocks perfectly. However there are several algorithms for synchronizing clocks approximately. All these algorithms have similar properties: relevant information is distributed, processes make decisions on local information, and single point of failure should be avoided. As an alternative to using physical clocks, logical clocks are used as a tool for distributing events without knowing exactly when they happened. The principle is that two events in a process have occurred in the sequence which was recorded by the process and when a message is sent between processes, the *send message* event will always happen before the *receive message* event. [C$^+$01, Ch 10] [LE03, Lec. 7] [WH03, Lec. 16] [Won03, Lec. 9]

Transparency The consequences of distribution in a distributed system are hidden by the systems transparency. It hides the separation of components and the system is perceived as one whole system instead of a collection of independent components. Transparency can be either at user, system manager, application programmer, or system programmer level, just depending on what is most efficient for the system.

There are many types of transparency which all are important for a distributed system. See figure 1.1 for relations between the transparencies. The figure is translated from slides by Olav Lysne. [LE03, Lec. 1]

Access Transparency: Access to local and remote components and resources by using the same operations is possible due to a system's access transparency. It hides the way a resource is accessed and various differences in data representation. Olav Lysne gives three examples of the use of access transparency: the file handling system-operations in Network File Systems, the navigation on the World Wide Web, and SQL-queries in distributed databases. [LE03, Lec. 1] Components without access transparency can not easily be moved from one machine to another. Coulouris et al gives an example of a lack of access transparency: "a distributed system that does not allow you to access files on a remote computer unless you make use of the ftp program to do so." [C$^+$01, p. 24]

Location Transparency: The location of resources is hidden by a system's location transparency. The users need not to know the exact physical location of the resources to access them. Olav Lysne gives three examples of the use of location transparency: the file handling system-operations in Network File Systems, the use of URLs on web pages on the World Wide Web, and tables in distributed databases. [LE03, Lec. 1]

Persistence transparency: Whether objects are located in memory or on disk should not be of any concern to applications or users. This is hidden by the persistence transparency. It should be regarded in close relation with location transparency.

Relocation transparency: An object, like resources or clients, may be moved from one part of the system to another without the need for changes in applications or user operations. This is due to the relocation transparency which works on resources in use. It hides the fact that a resource may be moved to another location while in use. An example of relocation transparency given by Coulouris et al [C^+01, p. 24] is the use of mobile phones. It does not matter for the person I talk to where I am as long as my mobile phone has contact with a base station. Relocation transparency is also known as migration and mobility transparency, and should be regarded in close relation with location transparency.

Concurrency Transparency: Users may access the same resource. Concurrency transparency organizes the resources in a hidden manner so the users are allowed to access resources without worrying about interference from others trying to use the same resources.

Replication Transparency: Like concurrency transparency, the replication transparency hides the fact that users may access the same resource. There may be multiple copies of a shared resource to make the resource sharing possible. An example of replication transparency given by Coulouris et al is the Domain Name System (DNS). Although it "allows a domain name to refer to several computers, it picks just one of them when it looks up a name." [C^+01, p. 24]

Failure Transparency: Failures may occur in a system, but the users should not be aware of those failures and recovery of such in the system. The failure transparency hides the failures and allows the users and applications to complete their tasks without interference. Coulouris et al gives an example of failure transparency by using electronic mail. If severs or communication links fail, delivering an email may fail. But the system will try to retransmit the messages, and the email will eventually be delivered, even though this may take several days. [C^+01, p. 24]

Performance Transparency: The work load of the system is hidden by the performance transparency. The system may keep the same performance level as earlier when the load increases. Sometimes reconfiguration is required, but this is hidden from the various users of the system by the performance transparency.

Scaling Transparency: The size of the system is hidden by the scaling transparency. If the size of the system changes, the systems structure and application algorithms are not affected.

The most important transparencies are the access and location transparency. They are sometimes referred to as one transparency called *network transparency*. If either one is absent it will reduce the utilization of the distributed resources. [C$^+$01, Ch 1.4] [LE03, Lec. 1] [WH03, Lec. 1] [Won03, Lec. 1]

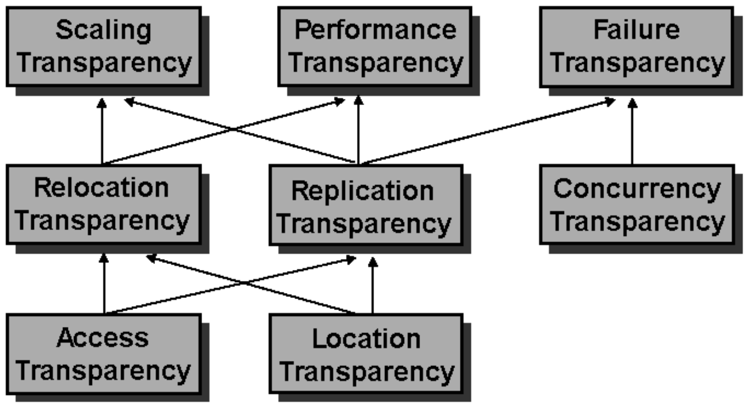


Figure 1.1: Transparencies

Security For most users of a system, security is very important, and it is likely that they will not use the system if they feel it is too insecure. This is especially important for systems that handle transactions where security and integrity are essential, for example financial transaction systems and military systems. The security issues that are of concern are confidentiality, integrity and availability. Or like Dr On Wong said it: "The main goal of security is to restrict access to information and resources to just those principals that are authorized to have access". [Won03, Lec. 12]

Security threats There are three classes of security threats: *leakage*, where unauthorized recipients get hold of information, *tampering*, where information is altered without authorization, and *vandalism*, where the perpetrator interferes with the operations of a system without any gain. The security in a distributed system is complex due to the need to secure information during transmission from one part of the system to another. Some of the threats to a distributed system in the various classifications are:

Denial of service: When a resource or message channel is flooded with messages so no others may use the channel, the channel is under a Denial of Service (DoS) attack. Lately the use of viruses has been a "popular" way of doing this. The perpetrator makes a virus that uses the affected machine to send messages to a particular site in the network. At the time of the attack, the system is so swamped with messages that it can no longer receive and interpret them. The system may then crash or fail in some other way. See figure 1.2.

Eavesdropping: A perpetrator may eavesdrop by listening to "conversations" between users of the system and obtain copies of the conversation without authorization. Information within a system should be protected from eavesdropping by unauthorized persons. This may include information on who is talking to whom as well as the matter itself. For military communication and businesses this is especially important. See figure 1.2.

Masquerading: To impersonate someone else without their knowledge, is called masquerading. This is sometimes used to gain access to other parts of the system than the authorization for the perpetrator allows or to gain secrets that other parties hold without their knowledge. It may also be used for discrediting the user being impersonated. See figure 1.2.

Message tampering: When tampering with a message, it is intercepted and altered before it is passed on to the intended recipient. The *man in the middle* attack is a type of message tampering where the perpetrator intercepts the first message in an exchange of encryption keys to establish a secure channel. The perpetrator then changes the message by substituting the key with her own before she sends the message to the intended recipient. This way the perpetrator will be able to listen to or take part in a conversation between other users without authorization and without the others knowledge. See figure 1.2.

Replaying: If a perpetrator stores intercepted messages, they can be used in a replay attack. Replaying attacks should be regarded in close relation with masquerading as they can be used for the same purpose. This type of attack may even be effective for encrypted or authenticated messages. See figure 1.2.

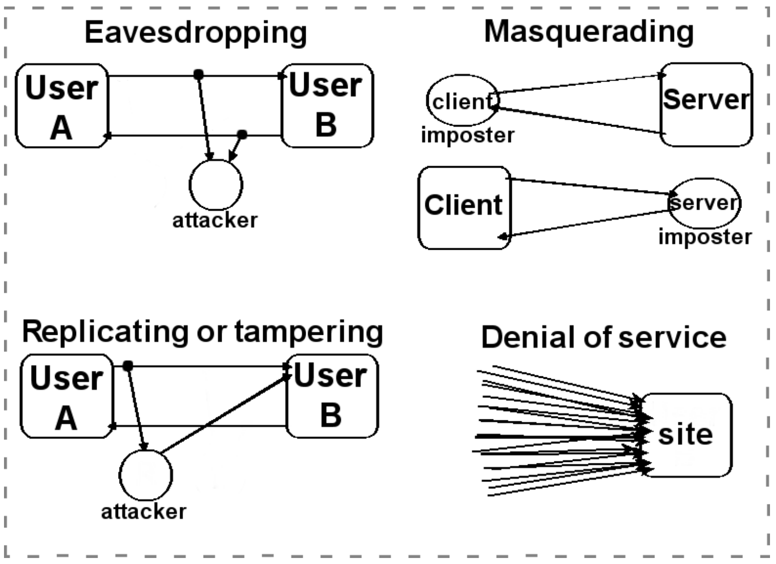


Figure 1.2: Security threats.

Security techniques When designing a distributed system, the security should be an important issue. One should design it as to avoid disasters and minimizing mishaps. One way to do this is to assume the worst case:
* Interfaces are exposed
* Networks are insecure
* Limit the lifetime and scope of each secret
* Algorithms and program code are available to attackers
* Attackers may have access to large resources
* Minimize the trusted base

Every distributed system should have a security policy. The security policy determines what should be done and by whom, and who have access to what and when. A policy can both be global for the whole system and local for parts of the system. Ian S. Welch lists eight points concerning global policy [WH03, Lec. 9]:
1. The environment consists of multiple administrative domains.
2. Local operations are subject to a local domain security policy.
3. Global operations require the initiator to be known in each domain in which the operation is carried out.
4. Operations between entities in different domains require mutual authentication.
5. Global authentication replaces local authentication.
6. Controlling access to resources is subject to local security only.
7. Users can delegate rights to processes.
8. A group of processes in the same domain can share credentials.

He also lists some points concerning local policy:
Discretionary: based on identity of requestor and access rules
Mandatory: based on mandated regulations determined by a central authority
Multilevel: prevents information flows down a hierarchy
Multilateral: prevents information flowing across a hierarchy

There are three widely used techniques used for security today: cryptography, authentication mechanisms and access control mechanisms.

Cryptography: Is used to conceal the message so only the intended communicating parties can understand it. It uses encryption and decryption of messages. The idea is to encrypt a message with a secret key, Ka, and send it to the intended recipient where the encrypted message is decrypted with a secret key, Kb. The secret keys Ka and Kb may be identical or they may be a pair. If the keys are complex enough and distributed between the two communicating parties in a secure manner, the original message is safe. There are several methods of cryptography, which I will not discuss here. For those who are more interested in the topic I will recommend the book by Nigel Smart. [Sma03]

Authentication mechanisms: Is used to make sure that all the communicating parties can be identified and that they are who they claim to be.

Certificates are very often used for this. They use cryptography by the use of private/public key. Here messages are encrypted with the private key Ka, and can only be decrypted by the use of the public key Kb. Since Ka is private, one can be sure that the sender is who she claims. I will also here recommend the book by Nigel Smart for those who are more interested in the topic. [Sma03]

Access control mechanisms: Is used to limit the access of the users to ensure that only the ones with the correct authorization enter specific locations. There are many types of access control. Most widely used are the Role Based Access Control (RBAC) and Access Control Lists (ACL). The RBAC divides users into groups based on their roles in a system, for example lecturers and students. Each group is then given certain access rights to parts of the system. The ACL gives access rights directly to each user. It is possible to combine these two access control mechanisms
[C$^+$01, Ch. 7] [Won03, Lec. 12] [WH03, Lec. 9-11] [Cro96, Ch. 4]

1.3 Specific Characteristics

Although there are many different types of distributed systems, there are few basic system patterns which the systems are based upon. The most familiar and widely used distributed systems are the Internet, intranets and networks based on mobile devices. These systems take advantages of other distributed systems like the naming service and the distributed file system.

Networks Distributed systems use networks for communication. The underlying networks have impact on the performance, reliability, mobility and quality of service of a distributed system. The hardware components of a network are composed of communication circuits, connections, routers, switches, interfaces, etc. The software components of a network are composed of protocol managers and stacks, communication handlers, drivers, etc. Some of the features of a network are:
Bridge: The link between two networks of different types.
Communication subsystem: The collection of hardware and software components which provides the facilities for communication.
Data transfer rate: The speed of transferring data between two components in the network, once a connection is established. The speed is measured in bits per second.
Gateway: The link from one network to another. It can be a router or other dedicated device.
Host: A device that uses the network; can be a computer or any other type of device.
Hub: A suitable way of connecting hosts.
Latency: The time it takes to send an empty message, the time it takes to

access the network at both ends. There are software delays at the sender and receiver, delays in accessing the physical network and delays within the network which all have effect on the latency.
Message transfer time: The time it takes to transfer a message. It is measured as: latency + data length / data transfer rate
Node: A device attached to a network, can be a computer or switching device.
Router: A link between two or more networks. It passes data packets from one network to another by using routing tables to get to a distant network.
Subnets: A collection of nodes which are located and can be reached on the same physical network, a unit of routing.
Switch: Similar as a router, but is only used for local networks.
Total system bandwidth: The total amount of data that can be transferred at any given time, it may involve more than one channel.

The performance of a network is given by the hardware and software used in the network. The two most important issues here are the latency and the data transfer rate, they affect the speed at which a message can be transferred between to components of the system. There is a variety of types of networks which are classified by size and usage. Se table 1.3 for comparison.

Local area networks (LANs): LANs are used within relatively small areas, they can contain as few as two computers. Direct transmission is used on single communication means, like twisted copper wire, coaxial cable or optical fibre. These can be connected by hubs or switches, but no routers are used. The speed used is relatively high, since high bandwidth is used and the latency is low except when the traffic is heavy. The technologies used here is usually Ethernet, token rings and slotted rings.

Metropolitan area networks (MANs): MANs can be used on areas up to 50 km and is usually used in a city or community. It has some of the same advantages as LANs, for example the speed is relatively high, as well as covering some of the aspects that used to be done by WANs. The network is based upon copper and fibre optical cables. The technology used is usually Ethernet and Asynchronous Transfer Mode (ATM). Good examples of a widely used MANs today are the ADSL and the cable modem connections.

Wide area networks (WANs): WANs are used on larger areas, up to thousands of km. The connection speed is lower than for LANs and MANs since the area is wider and the communication means are based on various technologies and have different bandwidth. The host computers are connected to the WAN by packet switches or packet switching exchange. The switches forward the packets to their destination and the transmission time depends on the route.

Wireless networks: Network connections to portable and handheld devices, such as laptops using wireless connection and mobile phones, are called wireless networks. Some of the networks are Wireless Local Area

Networks (WLAN) intended for use instead of wired LANs and can be connected at distances over 150 meters. Some are Wireless Personal Area Networks (WPAN) intended for connecting mobile devices to other mobile devices or connecting fixed devices in close proximity of another, these can be connected at distances just over 10 meters. Other Wireless Wide Area Networks (WWANs) are intended for usage across wider area. Mobile phones typically use this type.

Internetworks: Internetworks are communication subsystems. They rely on devices such as routers and bridges and allow for expansion with different network, link and physical layer protocols. The internetworks relies on a unified addressing scheme, on that the components are connected, and on a protocol defining the format of communication, the ones that exist today rely on the Transmission Control Protocol / Internet Protocol (TCP/IP).
[C$^+$01, Ch. 3] [Won03, Lec. 2]

	Range	*Bandwidth (Mbps)*	*Latency (ms)*
LAN	1-2 kms	10-1000	1-10
WAN	worldwide	0.010-600	100-500
MAN	2-50 kms	1-150	10
Wireless LAN	0.15-1.5 km	2-11	5-20
Wireless WAN	worldwide	0.010-2	100-500
Internet	worldwide	0.010-2	100-500

Table 1.3: Network comparisons. [C$^+$01, p. 72]

System patterns The system pattern or topology defines how the components in the system fit together. There are some basic patterns that are widely used, these are centralized, decentralized, hierarchical and ring systems, and various hybrids of these. See figure 1.3.

Centralized: This is probably the most familiar pattern. It is typically seen as the client/server pattern used by simple distributed systems. It is the historically most important and it is still the most widely used. In a centralized system there is one server where all functions and information are located. The clients connect to the server by sending and receiving messages in order to utilize the functions and information located there. The primary advantage of these types of systems is their simplicity. They are easy to manage and relatively easy to secure, since all data is concentrated in one place. The drawback is that if the central server crashes, the whole system breaks down. Hence, there is no fault tolerance. Another major drawback is lack of scalability. A centralized system can only be extended to a certain degree since the central server has a limitation.

Decentralized: This is a typical peer-to-peer (p2p) type of pattern. All the components communicate symmetrically and have equal roles. Many file sharing systems are designed as p2p systems. The primary advantage of a decentralized system is their extensibility. Any component may be added to any part of the system. However this makes it very difficult to manage. In addition, messages may carry a lot of overhead and the system may end up being slow and unpractical. Another advantage is the system's fault tolerance; it will not affect the whole system if just one node crashes.

Ring systems: A common solution to the problem of high client load is to use clusters of components arranged in a ring. A component communicates directly with only the two closest components, one on each side. To use this type of pattern, the components should be located at a close proximity of each other. These types of systems are like the centralized system, relatively easy to manage and to secure. It also has the advantage of being scalable; one can easily add another component without too much hassle. However there may be a problem with speed, as a message may need to pass through several components to arrive at the intended receiver.

Hierarchical: This pattern is similar to centralized, but here the central server may be a client in another centralized system in addition to its server capabilities in the system. The hierarchical systems use a tree-like structure. The primary advantage of the hierarchical systems is their scalability. A component can be added at any level. Although they are only partially fault tolerant in that if a server crashes, the clients below may easily be affected. They may also be somewhat hard to manage.

What type of system pattern one should choose for a distributed system, depends entirely on what kind of distributed system it is meant to be and what its usage is intended to be. Often a hybrid system is chosen, as there are almost no limits in how to combine the various system patterns. [Min01] [Min02] [C$^+$01, Ch. 2] [LE03, Lec. 2]

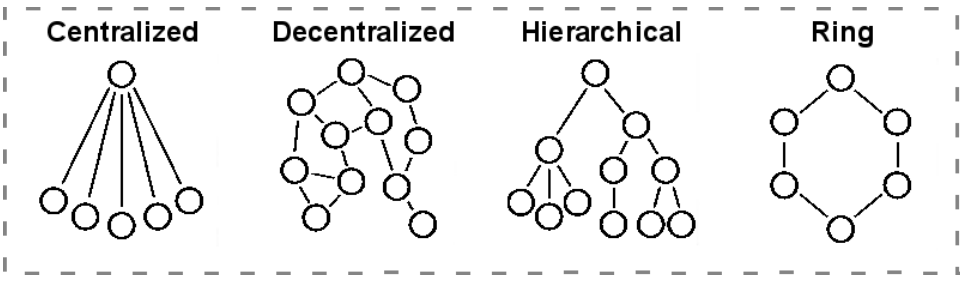


Figure 1.3: System patterns. [Min01]

Remote communication Some applications depend on cooperating programs running in other processes, often located on other computers. These applications need a way of communicating and invoking operations in those other processes. In order to achieve this some extensions have been made to familiar programming models and they now apply to distributed programs. These extensions all work on *middleware* layers. Middleware is a term used for "a software layer that provides a programming abstraction as well as masking the heterogeneity of the underlying networks, hardware, operating systems and programming languages." [C$^+$01, p. 16-17] See figure 1.4.

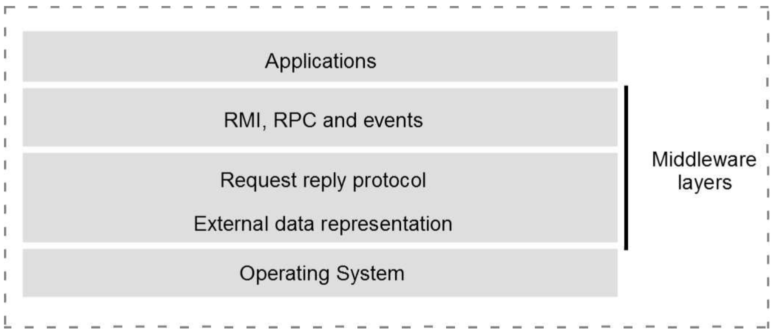


Figure 1.4: Middleware layer. [Won03, Lec. 4]

Remote procedure call (RPC): Procedure calls invoke procedures. They use the procedures interface which describes the procedures input, output, or both. Not all procedures return values. RPC allows a program to call procedures on another program running in a separate process. The program that issues the request is called a client and the responding program is called a server. The client and server rely on the use of request and reply messages, and very often they are running on different computers. See figure 1.5.

Remote method invocation (RMI): Object oriented programs consist of objects communicating with each other. The objects encapsulate their data and code of methods so the communication take place by invoking others methods. When dealing with distributed objects systems, the objects are managed by servers and the clients use RMI for invoking the servers' methods. RMI is similar to RPC since it is dealing with communication between different processes, very often running on different computers, but in RMI it is between objects instead of programs. And like RPC, the client and server rely on the use of request and reply messages. See figure 1.5.

Event-based programming model: An object may register its interest for particular events which may occur in other objects. The object then receives notification when such an event has occurred. In distributed event-based systems the same happens, but here it may also happen with remote objects. See figure 1.5.
[C$^+$01, Ch. 5] [LE03, Lec. 3] [WH03, Lec. 3 - 6] [Won03, Lec. 4]

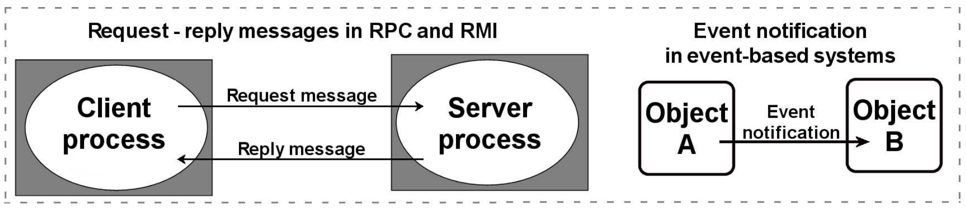


Figure 1.5: RPC, RMI and Event-based programming model.

Distributed programming Distributed applications run in distributed environments and make use of the characteristics of distributed systems. There are no restrictions to what programming languages the developers must use, but some languages and architectures have built in functionalities which may make the source code less complex.

CORBA: The Common Object Request Broker Architecture, CORBA, is an architecture specified by the Object Management Group. It is a powerful API for realising distributed systems of objects and it is programming language and operating system neutral. CORBA specifies how software objects are distributed over a network and how they can work together as clients and servers. The client uses the local CORBA Object Request Broker (ORB) to take care of the details of locating the objects, routing the requests, invoking the methods on the other object, and returning results. The client application then only needs to know the objects name and how to use the objects interface. The ORB is one of two system components and handles all the communication between the components. It lets the objects interact in a platform and implementation in a neutral way. The other system component is the object service. It performs the general object management tasks such as creating the objects and controlling access to the objects. The components of the application are the common facilities, which deal with configurable standard application functions, and the application objects deal with application domain functions and specific services.

Java: The Java programming language by Sun Microsystems has several functionalities for distributed programming, as this is one of the bases for its existence. Sun Microsystems declare that "The Java platform is the ideal platform for network computing." [Sun04b] The Java Interface Definition Language (Java IDL) is based on CORBA. It enables objects to cooperate in spite of differences in programming languages. The Java IDL provides an ORB, a class library, which enables CORBA-compliant applications to have a low-level communication with Java IDL applications. The Java 2 Platform Enterprise Edition (J2EE), a platform in the Java family, enables solutions for developing and managing multi-tier server-centric applications. The Enterprise JavaBeans (EJB) is the server-side component architecture J2EE. It encapsulates the business logic of an application.

DCOM: The Component Object Model (COM) is Microsoft Corpora-

tions framework for developing and supporting program component objects. COM objects are separate components with a unique identity. They publish their interfaces to allow applications and other components to access their features. Distributed COM (DCOM) is an extension of COM that allows the components to communicate in a distributed environment. The Active Template Library (ATL) is a library of template-based software routines. These can be used when creating COM and DCOM objects. COM+ is an extension of COM which adds a new set of system services for running application components. It is viewed as Microsoft's answer to the Sun Microsystems' EJB.
[Won03, Lec. 5-6] [IT03, Lec. 10-11] [C$^+$01, Ch. 5 +17] [Tec04] [Jup04]

Name service When communicating with a resource its name, address and attributes are important. Names are used for referring to resources. They may be local or stretch across the whole system. A name can be a textual identifier, such as human readable username, or a system identifier. Addresses refer to the location of a resource, not to the resource itself. When a resource changes its location, the address is changed as well. Attributes are values of properties associated with a resource. To use Coulouris et al's example: the Domain Name System (DNS), which is a global naming service whose principal naming database is used across the Internet, "maps domain names to the attribute of a host computer: its IP address, the type of entry (...) and, for example, the length of time the host's entry will remain valid." [C$^+$01, p. 355] When using general names there may be several names for one address or several addresses for one name, while unique identifiers refers to one single entry, each identity has at most one identifier, and the identifiers are never reused.

A name server provides clients with data about named objects in distributed systems. It stores a collection of one or more naming context. And its main function is to link a name and an attribute. The query may be by the name or on the attribute value or type, and it filters on all of these. The name server uses name spaces and name resolutions. A name space is a collection of the valid names recognized by a service. This means that it will try to look these up. It may use aliases to substitute a complex name with a more convenient one. A name resolution is a process where a name is continually presented to the naming contexts. The context then either maps the name directly on a set of attributes or on an additional naming context. When the name server queries for a name, if the name is not located in the first naming context, it will continue into the next, and so on until the name is found or there is no more contexts where to perform the query.
[C$^+$01, Ch. 9] [LE03, Lec. 5] [WH03, Lec. 13] [Won03, 10]

Distributed file system Persistent data are data that survive power outages. A file system stores this type of data. In a distributed file system persistent data is available across a network. The file services provide access to files stored at a server. If the file service is well designed the performance and reliability is similar or even better than with files stored on a local disk. The users may access shared files from any computer in a network since the "distributed file system enables programs to store and access remote files exactly as they do local ones." [C^+01, p. 309] Consequently the users may also perform read and write operations. The users do not necessarily know that the files are remotely stored, since this is transparent through the distributed file system. Disks are relatively slow, so files recently accessed are typically cached in memory. Whether to have the caching at the server, at the client or both places usually depends on usage of the file system. The best examples of distributed files systems are the Network File System (NFS) from Sun, the Andrew File System (AFS) developed at Carnegie Hall, and the Digital Multimedia Server (DMS). [C^+01, Ch. 8] [Won03, Lec. 1] [Cro96, Ch. 10]

Internet The Internet is one of the most typical distributed systems, although people may not generally think of it as one. Some think of the Internet as just the World Wide Web (www), although this is just one of the enabled services. Other services, such as email and files transfer, are widely used and just as important. The Internet is a world-embracing set of computer networks, all linked together as one. See figure 1.6 for a typical portion of the Internet. This figure shows a set of intranets (description in the paragraph below). These are subnetworks to the Internet network and are operated by companies or organisations. ISP is an abbreviation for Internet Service Providers, which are companies that provide Internet access to individual users and small companies or organizations. The backbones are network links; they have high transmission capacity and are based upon fibre optic cables, high bandwidth circuits, and satellite connections. Many of the distributed systems and distributed applications today communicate through the Internet and the various services it provides. [C^+01, Ch. 1]

Intranet The intranets are, as described above, subnetworks to the Internet network and are operated by companies or organisations. They have boundaries towards the Internet and are able to enforce local security policies. Typically an intranet is composed of several LANs. These are linked together through backbones just as the Internet. The intranet is then connected to the Internet through a router. The router allows the users inside to make use of various services outside of the intranet as well as limiting the access to the intranet from the outside. The router then acts

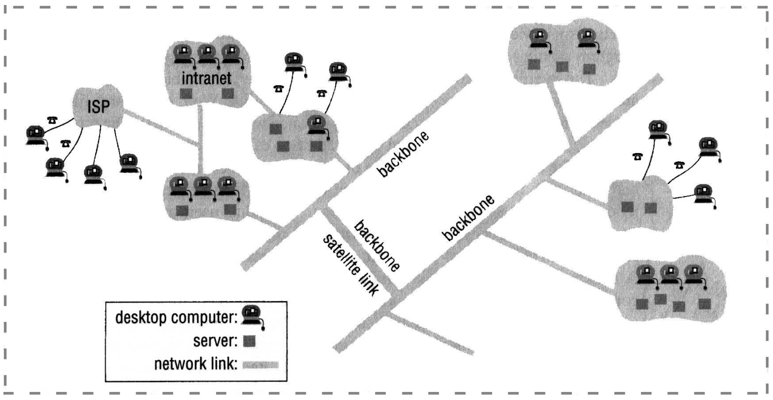


Figure 1.6: Typical portion of the Internet. [C$^+$01, p. 3]

as a firewall. A firewall prevents unauthorized messages from entering or leaving by filtering the messages, for example by their source or destination address. Some organizations may not even connect their intranet to the Internet at all, for example military constellations and hospitals. These intranets have the same infrastructure as other intranets except for the router/firewall. Intranet may be of various sizes they may range from large ones in large companies, consisting of hundreds of computers, to small ones in private homes, maybe consisting of no more than two computers. See figure 1.7 for a typical intranet. [C$^+$01, Ch. 1]

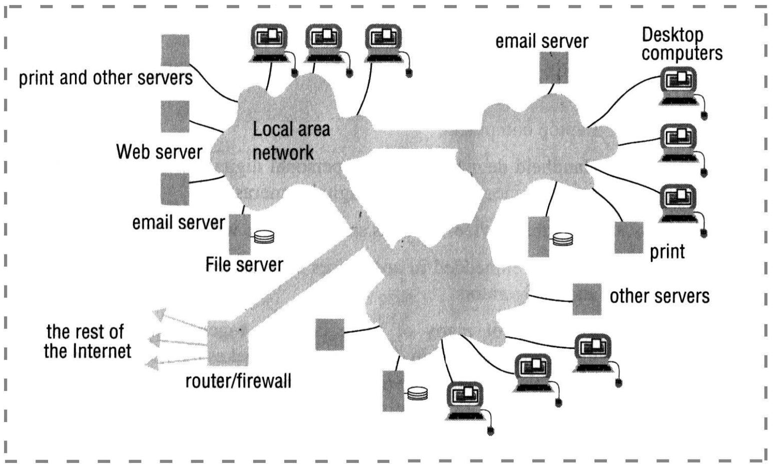


Figure 1.7: Typical intranet. [C$^+$01, p. 5]

Mobile devices The advances in technologies of device miniaturization and wireless networking have helped in increasing the interest and use of small and portable computing devices. Today it is very common to own

and use at least one of these devices, such as
* laptop computers;
* handheld devices, like mobile phones, pagers, personal digital assistants (PDAs), and digital or video cameras;
* wearable devices, like smart watches;
* devices embedded into appliances, such as refrigerators and cars.
These devices are called mobile devices because of their ability to connect to networks in different places as well as their portability. These devices can move between various technology environments with differences in bandwidth, latency, loss, etc. Some of these devices are present and so closely linked to a user's physical environment that she may hardly notice it, so-called ubiquitous devices. See figure 1.8 for example of portable and handheld devices in a distributed system. The figure illustrates the home intranet and the visiting site of a user who is visiting a host organization. The user accesses three types of wireless connection: the laptops connection to the hosts wireless LAN, the mobile phone using the Wireless Application Protocol (WAP) to connect to the Internet, and a digital camera which communicates with a printer using an infra-red link. The wireless LAN in this network would usually cover a few hundred meters, typically the floor of a building, and would be connected to the rest of the host's intranet through a gateway. [C+01, Ch. 1]

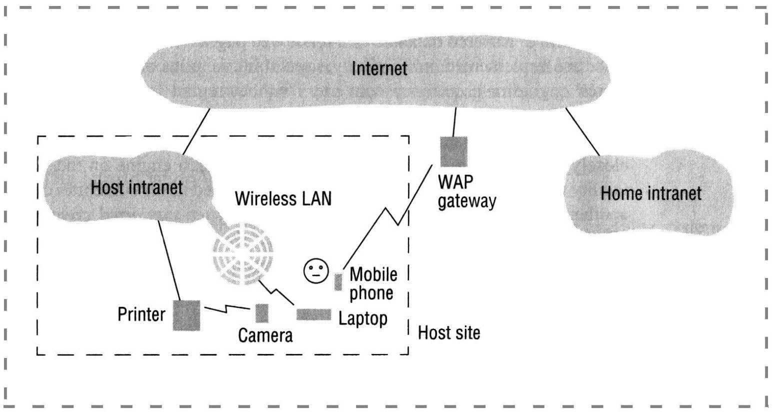


Figure 1.8: Portable and handheld devices. [C+01, p. 7]

Chapter 2

Web Services

2.1 Introduction

In this chapter web services are described. I have used the book by Graham Glass [Gla01] as a basis for the chapter, and it is used as a reference where nothing else is indicated. I will recommend this book to those who are interested in learning more about web services and to those who want to start to develop their own web services.

Web Services is a relatively new development. It is based upon the principles of distributed systems. And like in distributed systems, the components communicate with each other only by passing messages. Many people regard web services as a technology only for publishing software services on the Internet via browsers, while others regard them as the "new big thing" in distributed computing that is working as general purpose architectures.

The World Wide Web Consortium (W3C) describes web services and the interaction between the components: "A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards." [W3C04c]

Before web services were introduced, information was exchanged through presentations. A presentation was then made in Hypertext Markup Language (HTML) or some other presentation language and applications on the web were made in a pre-chosen technology. The applications could rarely interact because of limitations in these technologies.

Web services typically use standard network protocols like Hypertext Transfer Protocol (HTTP) for transmitting messages and Extensible Markup Language (XML) as format for the message content. HTTP is an application protocol which runs on top of TCP/IP (regarded under networks in section 1.3).

The use of web services is increasing rapidly in accordance with the increase of development and use of application-to-application communication and interoperability. Web services provide standards for communication between applications over a network. I will try both to describe these standards as well as highlight the general characteristics of web services.

2.2 General Characteristics

A web service is a set of functions that is published to a network for use by other programs. This is possible by enclosing the functions into one single entity, the web service. The communication is done only by passing messages and enables the programs and web services to be implemented on any platform and in any programming language.

The web service architecture models the communication between the software components. It is based upon exchanging messages between the component that provides the web service (service provider), the component that requests the web service (service requestor), and the component where the information about the service is published (discovery agency). The service provider publishes descriptions of each service it provides to a discovery agency. The service requestor then obtains the description of a desired service from the discovery agency and sends a request to the service provider based on this description. The service provider executes the service and sends the appropriate response to the service requestor. See figure 2.1.

Challenges and Concerns In order for web services to be successful, there are some technical challenges which need to be met and some concerns which should be diminished.

Quality of Service: Availability and performance are some of the concerns when using web services. A web service is based on XML, which is text-based. Hence it entails more data for the systems to process than if it were binary-based. This can cause the web service to run relatively slowly over the HTTP. If additional security protocols are added (see paragraph below), the web service would run even slower. Another concern is the quality of a web service. Today a lot of the publicly available web services are open source; hence the developers are not paid. The end users may therefore be used to getting services for free, and will not be willing to pay for the service to be enhanced or upgraded. These versions may be beta-versions with much raw code and there is no support service if needed. There is also the concern about reliability when using an externally made web service. It may be difficult to know which host is reliable, whether it is

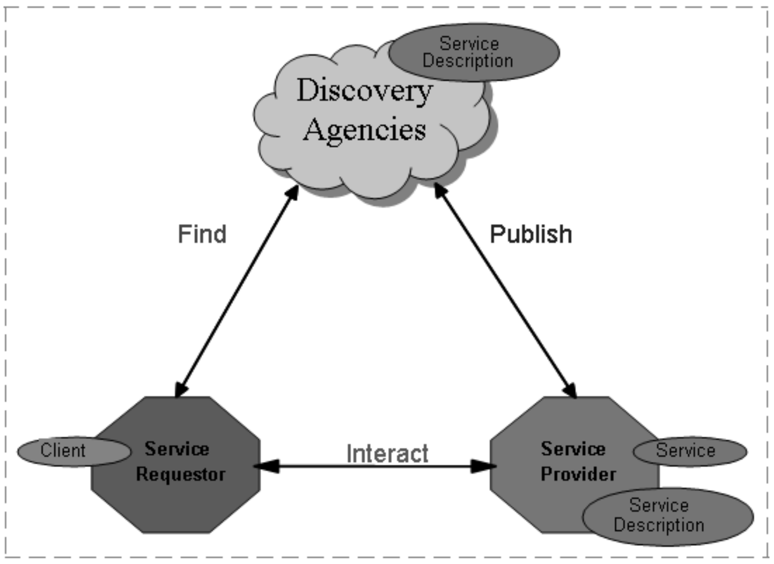


Figure 2.1: Web service architecture. [W3C04b, Nov. 2002]

when distributing and advertising web services or when locating and using them. A system which involves several externally made web services, may collect them from different vendors. And they may be hosted in different environments and on different operating systems. Such a system may not be as predictable as desired. Due to this, challenges concerning testing, debugging, and performance may arise.

Scalability: It is possible to expose existing component systems as web services, many of which are scalable. It may be a challenge to preserve the desired scalability in the web services as well.

Security: Privacy and security are potential sources of concern in any computer system. This is treated more closely in the paragraph about security (see below).

Transactions: Many traditional systems use the two-phase commit control approach in transactions. Here all the participating resources are gathered and locked until the transaction is finished. This is unpractical in open environment systems where transactions can last for hours or days, such as in systems which use web services. IBM developerWorks specifies two types of web service transactions: Atomic transaction (AT) and Business activity (BA). ATs are used on transactions with short duration and limited trust domain. They are similar to the transaction specification in the distributed systems chapter (section 1.2). BAs are examples of compensating transactions. These transactions have two scenarios: A normal scenario which performs the operations specified by the transaction and a compensating scenario which performs operations to remove the effects of the normal scenario if this is required.

[Gla01, Ch. 1] [dev02c] [VN02] [Lou01] [Roe03] [Jen01]

Benefits The web service standards (see section 2.3) are relatively young standards and there are still some challenges (see above) that needs to be overcome. Even so web services are greatly used in today's business-to-business web-based solutions. The main reason for this is their many benefits:

Discrete: Web services support a more loosely coupled architecture than most traditional system architectures. And each web service typically only provides a single piece of functionality and is completely independent. Because of this an application can be broken down into a set of other web services. Each of these can in turn be offered as an independent web service.

Easy to implement, understand and use: Developers may use the component object model, architecture, implementation strategy, and programming language of their own choice, as long as they respect the web service standards. This makes it possible for a developer to make a web service without previous knowledge of the target system's environment. And web services based on different languages can be incorporated into a system and communicate with each other without problems. It is easy to find a web service and include it in an application. This is due to the publication of the web service's description and because they are independent

Industry support: Almost all major hardware and software vendors support the web service standards, ensuring that components may easily be deployed as web services or consumed by them. For instance the Microsoft .NET platform is based on web services. And because a web service access and communication happens in real time, data can be immediately updated. This ensures data integrity at all times.

Interoperable: A web service can interact with any other web service. This is because they communicate only by sending messages, in XML format. Hence the developers need not worry about what programming language or platform other web services are based on when designing a web service that may communicate with these. A web service can run on any kind of machine and with any kind of platform that supports web services. This includes rather small hand held devices.

Reusable: A web service can be extended and reused whenever necessary. Thus a developer does not need to make a web service from scratch when it should be extended. The developer may extend her existing web service by adding the desired functionality. In addition a web service can incorporate existing systems and applications in order to make them accessible and usable in new areas and systems.

Ubiquitous: Since web services usually use HTTP for communicating over a network and XML as format for the communication, any device which supports these technologies can access or host a web service. In addition, web services respect existing security systems, because of the development standard, and can therefore use the existing infrastructure at the time and location of access.

Understandable: Web services are understandable for both humans and computers. A human can for example understand a web service through an application while the computer understands the same web service through an Application Programming Interface (API).
[Gla01, Ch. 1] [Lou01] [W3C03c, v. Mar. 2003] [VN02] [Cap01] [Epi03]

Security There are two ways of securing a web service, by using existing external security measures or the web service security standard.

External measures: There are several external security measures that may be used with web services. `Transport security`: Existing technologies such as Secure Socket Layer (SSL) and Transport Layer Security (TLS) provide a simple point-to-point integrity and confidentiality for a message during transport. TLS is the introduced successor to SSL and it "is a protocol that ensures privacy between communicating applications and their users on the Internet. When server and client communicate, TLS ensures that no third party may eavesdrop or tamper with any message." [Tec04] `Public key infrastructure (PKI)`: Enables users of a traditionally insecure public network to exchange data in a secure and private manner. "At a high level, the PKI model involves certificate authorities issuing certificates with public asymmetric keys and authorities which assert properties other than key ownership (for example, attribute authorities). Owners of such certificates may use the associated keys to express a variety of claims, including identity." [CC02] `Kerberos`: Makes possible the secure authentication of requests for services in a computer network. "The Kerberos model relies on communication with the Key Distribution Center (KDC) to broker trust between parties by issuing symmetric keys encrypted for both parties and "introducing" them to one another." [CC02]

Web service security standard (WS-Security): The WS-Security standard is an industry standard from IBM, Microsoft, and VeriSign [dev02b]. It describes enhancements to the SOAP messaging (See subsection 2.3.2) and uses the other security measures as a design basis. It intends to protect the integrity and confidentiality of a message and authenticating the sender. The standard also specifies how to associate an unspecified security token with a message and how to encode certificates and Kerberos tickets. Donald Flinn proposes the use of the WS-Security as countermeasure to the following threats [Fli03]:
Un-authenticated sender - Use tokens and digital signature
Unauthorized receiver - Use XML encryption

Replay - Digital signatures alone are not enough to defeat replay. Other parts of the specification must be used with d-sig, such as timestamp, sequence number and nonce.
Token Substitution - Sign both the security header and the body.
Message modification - Sign the message
Message substitution - Sign both the security header and message body
Man-in-the-middle - Sign both the request and response
Multiple tokens using the same key - Require that the token be included in WS-Security header.
See figure 2.2 for overview of web services security specifications. [Gla01, Ch. 5] [dev02b] [dev02a] [CC02] [Tec04]

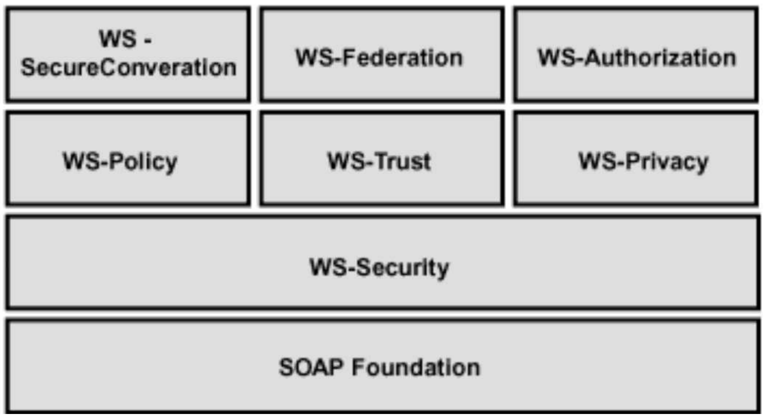


Figure 2.2: Web services security specifications. [CC02]

2.3 Specific Characteristics

A web service consists of four basic elements: global discovery, metadata, encoding, and transport (See table 2.1). As stated earlier, web services typically use existing transport protocols like HTTP and XML is the standard way of representing data.

Global Discovery	**UDDI**: Universal Description, Discovery and Integration `www.uddi.org` (also ebXML)
Metadata	**WSDL**: Web Service Definition Language (XML) `www.w3.org/2002/ws/`
Encoding	**SOAP**: XML encoded messaging / RPC `www.w3.org/2002/ws`
Transport	HTTP / HTTPS

Table 2.1: Web service overview. [Roe03, Lec. 11]

2.3.1 Data representation

Datatype, message format, and structure specifications should be based on the specifications of structures and datatypes in the W3C XML specification. However, other schema languages, such as RELAX NG and DTD may also be used. [W3C03c]

XML Extensible Markup Language (XML) is a descriptive meta-language. It is used to define the structure of documents and the names of the attributes. XML documents uses tags in data items to identify the data and attributes, and if their names are well chosen, to define their meaning. The document is composed of storage units called entities designed in a tree structure, beginning with a root or "document entity". The document is logically composed of character references, comments, declarations, elements, and processing instructions. The W3C defines that "Each XML document contains one or more elements, the boundaries of which are either delimited by start-tags and end-tags, or, for empty elements, by an empty-element tag. Each element has a type, identified by name, [...], and MAY have a set of attribute specifications." [W3C04a] An attribute is defined by its name and value. XML is designed to be directly usable over the Internet and to support a variety of applications. XML documents are human readable and easy to create, it is also easy to write programs to process them. See figure 2.3 for a simple example of the XML structure. [W3C04a] [W3C04b] [Edm02, Lec. 4] [Fer02]

```
<Phonebook>
    <Entry>
        <Name>
            <Title>Mr</Title>
            <LastName>Ola</LastName>
            <FirstName>Normann</FirstName>
        </Name>
        <Adress>Somecity</Adress>
    </Entry>
    <Entry>
        <Name>
            <Title>Miss</Title>
            <LastName>Jane</LastName>
            <FirstName>Doe</FirstName>
        </Name>
        <Adress>Somehwere</Adress>
    </Entry>
    ⋮
</Phonebook>
```

Figure 2.3: XML

DTD Document type definition (DTD) may be used as the schema language for WSDL (subsection 2.3.3). It can not be embedded, hence it must be imported, and a namespace must be assigned. The W3C defines DTDs as grammars for a class of documents. The grammars are composed of markup declarations which are declarations of element types, attribute lists, entities or notations. The DTDs may point to external declarations or contain the markup declarations, or both. [W3C03c] [W3C04a]

RELAX NG A RELAX NG schema may, like DTDs, be used as the schema language for WSDL (subsection 2.3.3). It can be either embedded or imported, but imported is preferred. And like for DTDs, a namespace must be specified. RELAX NG is a simple schema language for XML and is itself an XML document. It specifies patterns for content and structure of an XML document. [W3C03c] [OAS01]

2.3.2 SOAP

Simple Object Access Protocol (SOAP) is the messaging protocol for web services. It is used to encode the information in the request and response messages. A SOAP message is operating system independent and it is a way for one program to communicate with other programs independently to the operating systems of the other programs. The SOAP messaging framework consists of four parts, the SOAP processing model, the SOAP extensibility model, the SOAP protocol binding framework, and the SOAP message construct. The processing model defines "the rules for processing a SOAP message". [W3C03b] The extensibility model defines "the concepts of SOAP features and SOAP modules". [W3C03b] The underlying protocol binding framework describes "the rules for defining a binding to an underlying protocol that can be used for exchanging SOAP messages between SOAP nodes". [W3C03b] And the message construct defines "the structure of a SOAP message". [W3C03b] When an application receives a SOAP message, it must process that message. This is done by identifying all parts of the message that are intended for the application and verifying that all mandatory parts identified are supported by the application and process them accordingly. The message is discarded if not all mandatory parts are supported; any unsupported optional parts are ignored. If the message is to be forwarded, the parts identified are removed before forwarding the message.

SOAP extensibility model The core functionality of SOAP deal with providing extensibility. The extensibility model gives two means of expressing features: the SOAP processing model and the SOAP protocol binding model (both described below). The specification of a feature must include: a URI to name it, the information or state required to implement it, information about the processing required to fulfil the features obligations, and the information to be transmitted from node to node. A Message Exchange Pattern (MEP) is a type of feature in a template form which establishes a pattern message exchange between nodes. A SOAP module can realize several features. It is a specification of the semantics and syntax of header blocks. [W3C03b] [Nat03]

SOAP processing model This model describes what actions a SOAP node should take on receiving a SOAP message. A node can be the initial sender, the ultimate receiver or an intermediary, and is identified by a Uniform Resource Identifier (URI). The URI contains certain node attributes. The optional role attribute is to be played by the intended target of the header block. There are three standard roles: none, next, and ultimate receiver (see table 2.2 for description). The `mustunderstand` attribute is used to ensure that the nodes do not ignore important header blocks, and is set to `true` if the block must be processed. The `relay` attribute indicates whether a header block targeted at intermediary nodes must be relayed if not processed. [W3C03b] [Nat03]

Short-name	**Name**	**Description**
next	"http://www.w3.org/ 2003/05/soap-envelope/ role/next"	Each SOAP intermediary and the ultimate SOAP receiver MUST act in this role.
none	"http://www.w3.org/ 2003/05/soap-envelope/ role/none"	SOAP nodes MUST NOT act in this role.
ultimateReceiver	"http://www.w3.org/ 2003/05/soap-envelope/ role/ultimateReceiver"	The ultimate receiver MUST act in this role.

Table 2.2: SOAP Roles defined by this specification. [W3C03b]

SOAP protocol binding framework is a specification of how messages can be passed from one node to another using an underlying protocol. It provides rules for the specification of protocol bindings and descriptions of the relationship between the bindings and the nodes which implement these bindings. There are several types of bindings that can be used. A feature which is not available through a binding may be implemented using header blocks containing SOAP modules. According to the W3C [W3C03b], a SOAP binding specification has the following features; it:
* Declares the features provided by a binding.

* Describes how the services of the underlying protocol are used to transmit SOAP message infosets.
* Describes how the services of the underlying protocol are used to honor the contract formed by the features supported by that binding.
* Describes the handling of all potential failures that can be anticipated within the binding.
* Defines the requirements for building a conformant implementation of the binding being specified.
[W3C03b] [Nat03]

SOAP message construct The structure of a SOAP message includes an envelope, a header, a body, and a fault definition. See figure 2.4 for a structure overview.

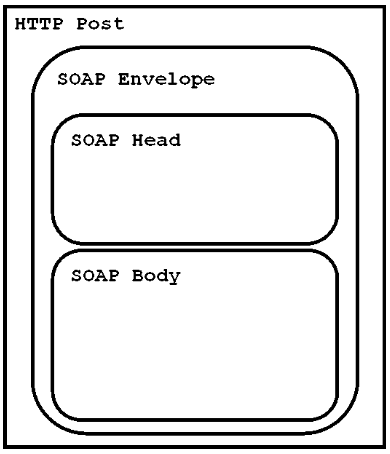


Figure 2.4: SOAP structure overview. [Nat03]

The envelope has the element name `Envelope` and it is the top element in the message. It is a mandatory part of the message which defines the overall framework of the message. The framework expresses what a message is, who should deal with it and if it is optional or mandatory. The envelope may contain additional attributes and namespace declarations. The attributes may contain sub elements. Both the attributes and its sub elements must be namespace qualified.

The envelope's first child in a hierarchic structure is the header. It is an optional element in the message and its element name is `Header`. Its function is to add features to a message without prior agreement between the communicating parties. The header element is identified by its element name, which consists of a namespace URI and a local name. All its children must also be namespace-qualified. The header has a few attributes that can be used to specify what role (see paragraph about SOAP processing model) should deal with a feature and whether it is optional or mandatory.

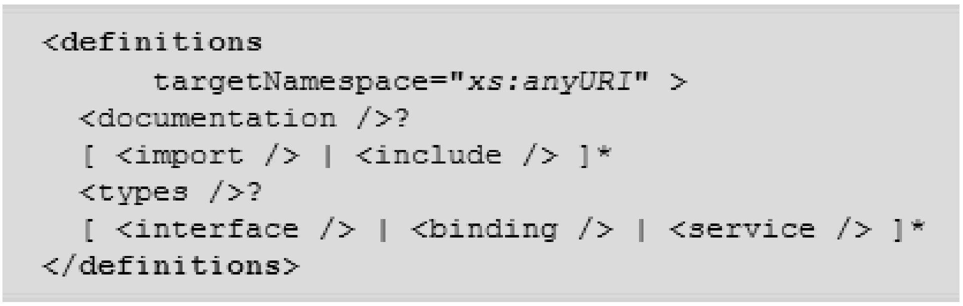
The other child of the envelope is the body. It has the element name `Body` and is a mandatory element. It is either located as the first child of the envelope or as the second child if the header element is present. The body may contain children as a set of body entries. These children may be namespace-qualified. The Body contains for mandatory information intended for the ultimate recipient of the message.

The fault element is used for indicating and reporting errors within the message. Its element name is `Fault`. The fault element contains two or more child element information items to describe the fault. The code and reason items are mandatory, the node, role, and detail items are optional. An element information items may appear within a header block or as a descendant of a child within the body, then the element has no SOAP-defined semantics.
[Won03] [W3C04b] [W3C03a] [W3C03b] [Edm02, Lec. 9] [YS02]

2.3.3 WSDL

Web Services Description Language (WSDL) is an XML language for describing web services. WSDL describes a web service at both an abstract and a concrete level. At the abstract level, the web service is described by the messages it sends and receives. Typically an XML Schema is used for this. At the concrete level, the web service's transport and wire format for one or more interfaces are described. Both the WSDL components and type system components are to be described.

Component model The conceptual model for WSDL is described as a set of mandatory or optional elements with properties. The W3C describes these elements as components. See figure 2.5 for structure overview.



```
<definitions
      targetNamespace="xs:anyURI" >
  <documentation />?
  [ <import /> | <include /> ]*
  <types />?
  [ <interface /> | <binding /> | <service /> ]*
</definitions>
```

Figure 2.5: WSDL structure overview. [W3C03c]

Definitions: This is the top level component and is similar to the envelope in the SOAP structure. At the abstract level, this component is just a container for the WSDL components and type system components. The WSDL components are interfaces, bindings and services. The type system components are element declarations and type definitions for some type systems. There are several properties of the definitions component: a local name (`definitions`), a namespace name, one or more attributes, and zero or more element information items. The attributes includes the mandatory `targetNamespace` as well as other namespace qualified attribute information items. The `targetNamespace` which is a logical namespace for information about the service. It is a convention of XML Schema and enables the WSDL document to refer to itself. Element information items are children of the definitions component and include `interface`, `binding`, `service`, `types`, `include`, `import`, and `documentation` which are described below, as well as other namespace-qualified element information items.

Interface: This component describes collections of messages a service sends or receives. Related messages are grouped into operations where an operation is a group of input and output messages. An interface is a set of operations and can optionally extend to one or more interfaces. An interface is defined by the following properties: a local name (`interface`), a namespace name, one or more attributes, and zero or more element information items. Attributes include a mandatory `name` and the optional `extends` and `styleDefault` as well as other namespace qualified attribute information items. The `name` together with the `targetNamespace` defined in definitions identifies the interface. The `extends` lists what interface this one is derived from. The `styleDefault` defines the default style used to construct the message. Element information items include `operation`, `feature` and `property`, as well as other namespace-qualified element information items. The `operation` is a collection of interface operation definitions. The `feature` is a collection of feature definitions. The `property` is a collection of property definitions.

Binding: This component describes a concrete message format and transmission protocols that may be used to define an endpoint (see under service below in this paragraph). The component may describe the information in a specific manner for a specific interface or in a general manner for any interface. The component may also define operation specific binding details, but then the component must define an interface. A binding is defined by: a local name (`binding`), a namespace name, one or more attributes, and zero or more element information items. Attributes include a mandatory `name` and an optional `interface` as well as other namespace qualified attribute information items. The `name` together with the `targetNamespace` defined in definitions identify the binding. The `interface` refers to an interface component. Element information items include `operation`, `feature` and `property`, as well as other namespace-qualified element information items. The `operation` is a collection of binding operation definitions. The `feature` is a collection of feature definitions. The `property` is a collection of property definitions.

Service: This component describes a set of endpoints at which the interface of the service is provided. The endpoint specifies the location for accessing the service. There are several properties of the definitions component: a local name (`service`), a namespace name, two or more attributes, and one or more element information items. Attributes include a mandatory `name` and a mandatory `interface` as well as other namespace qualified attribute information items. The `name` together with the `targetNamespace` defined in definitions identifies the service. The `interface` refers to an interface that the service is an instantiation of. Element information items include an optional `documentation` and the mandatory `endpoint` as well as other namespace-qualified element information items. The `documentation` element information item is described below in this paragraph. The `endpoint` is a set of endpoint components containing at least one endpoint component.

Types: This component is an optional component to declare all the data types the service uses that are not built-in the service, like arrays and structures. It is a collection of imported and embedded schema components, of any schema language. A type is defined by the following properties: a local name (`types`), a namespace name, zero or more attributes, and zero or more element information items. Attributes may be any namespace qualified attribute information items. Element information items include an optional `documentation` as well as other namespace-qualified element information items. The `documentation` element information item is described below in this paragraph.

Include: This component is a mechanism to help make the WSDL descriptions clearer. It allows for the separation of various components of a service definition, which originate from the same target namespace, into independent WSDL documents. These documents can later be merged as needed. An include component is defined by: a local name (`binding`), a namespace name, one or more attributes, and zero or more element information items. Attributes include a mandatory `location` as well as other namespace qualified attribute information items. The `location` is the location of the relevant information. Element information items include an optional `documentation` as well as other namespace-qualified element information items. The `documentation` element information item is described below in this paragraph.

Import: This component is like `include`, a mechanism to help make the WSDL descriptions clearer. It also allows for the separation of various components of a service definition into independent WSDL documents. But in this case the components have different target namespaces, which can be imported when required. There are several properties of the definitions component: a local name (`import`), a namespace name, one or more attributes, and zero or more element information items. Attributes include a mandatory `namespace` and an optional `location` as well as other namespace qualified attribute information items. The `namespace` "indicates that the containing WSDL document MAY contain qualified references to WSDL definitions in that namespace." [W3C03c] The `location` is the location of the relevant information. Element information items include an optional `documentation` as well as other namespace-qualified element information items. The `documentation` element information item is described below in this paragraph.

Documentation: This component contains human readable or machine processable documentation, or both. The documentation component is defined by: a local name (`documentation`), a namespace name, zero or more attributes, zero or more child element information items, and zero or more character information items.
[W3C03c] [Gla01, Ch. 3]

2.3.4 UDDI

If a web service is to be used, the information about how to access it must be published. Universal Description, Discovery, and Integration (UDDI) allow a web service's access information, like location, WSDL, and owner, to be published. The main purpose of UDDI is to provide an API for publishing and discovering information about a web service. UDDI specifies a framework that "will enable businesses to:
* Discover each other
* Define how they interact over the Internet
* Share information in a global registry that will more rapidly accelerate the global adoption of B2B eCommerce " [Mac03]

The UDDI technology "is a layer on top of standards-based technologies such as TCP/IP, HTTP, XML, and SOAP to form a uniform service description format and service discovery protocol." [Mac03] See figure 2.6 for the relationship between UDDI and these other technologies.

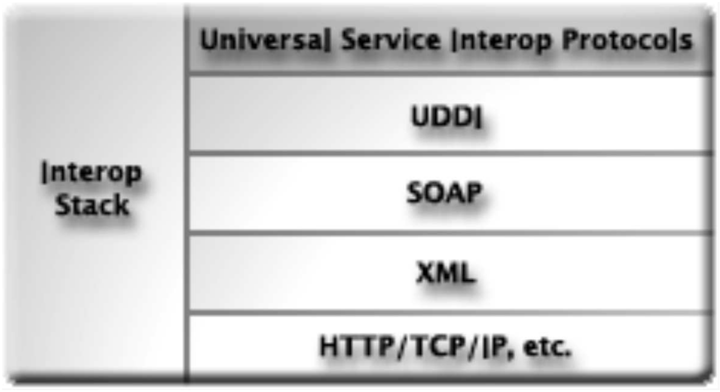


Figure 2.6: UDDI technology overview. [Mac03]

UDDI types There are three types of UDDI: public, protected, and private.

Public: This is probably the most known type of UDDI. Here anyone can access the UDDI registry to publish web service and business information or retrieve information about other web services and businesses. Access to the registry is typically free. Companies that want to publish general-purpose web services use this UDDI type.

Protected: These UDDI registries are typically industry specific and only accessible to certain businesses. The UDDI servers are often owned by an industry consortium and only the members of the consortium may access the registry due to performance and security reasons.

Private: Some companies have an internal UDDI server and registry. Here web services for internal use are published.
[Gla01, Ch. 6] [Mac03]

Data types The UDDI registries store information divided into various types. These are called entities. The entities are stored in a data registry similar to a phone book, which is split into White Pages, Yellow Pages, and Green Pages. There is also a section for Service Type Registration, service types are defined by tModels. See figure2.7 for overview of the registry datatypes.

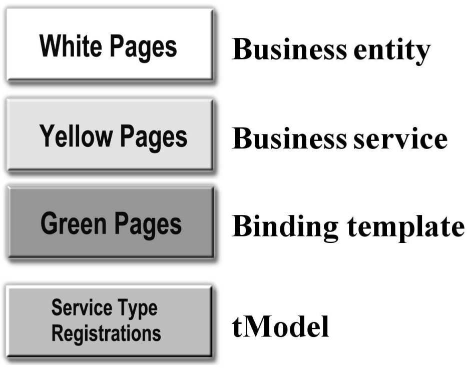


Figure 2.7: Registry datatypes. [Roe03, Lec. 11]

Business entity: Information about the business or organization that provides the web service. The entity is searchable in the UDDI White Pages. A business entity contains business services.

Business service: Descriptive information about a collection of related web services that are offered by the business entity. The entity is searchable in the UDDI Yellow Pages. A business service contains binding templates.

Binding template: Technical information about a single web service. It contains all the information necessary to locate and invoke the web service, including the URL of the web service, an optional description, and tModel references. The entity is searchable in the UDDI Green Pages.

tModels: Technical specification about a web service. The tModel has a name, a unique key, an overview URL to associated data, and may have sets of descriptions. The main purpose of a tModel is to represent a WSDL interface type.

Publisher assertion: This is an optional entity. It describes a business entity's relationship with another business entity, from the first business entity viewpoint. The publisher assertion may be relevant if a business is not effectively represented by a single business entity.

Subscription: This is also an optional entity. It is for keeping track of changes to entities described by the subscription.
[Gla01, Ch. 6] [Mac03] [UDD03]

Chapter 3

Practical

In this chapter I will look at the development aspect of both distributed systems and web services. I have tried to develop both a distributed and a web service application for the same problem. I have chosen two problems which are more closely described in the sections below.

In the poker problem, I have developed a distributed and a web service application from a non-distributed application. In the bulletin board problem, I have developed a web service application based on an existing distributed application. The source code, SOAP and WSDL documents to all applications are on the enclosed CD.

All applications are written in TextPad 4.7.2 [Hel] and compiled by Java^{TM} 2 SDK Standard Edition v 1.4.2 [Sunb]. The platform was Microsoft Windows XP Professional [Mic]. For help I have used the Java^{TM} 2 SDK Standard Edition Documentation v 1.4.0 and 1.5.0 [Sunb], especially the API specifications, Java programs I have previously made, and searched on `www.google.com` for help if needed.

In the web service applications I also used the server Apache Tomcat 5.0.19 from the Apache Jakarta Project [Apaa] and the Apache Axis 1.1 technology from the Apache Web Services Project [Apab]. The documentation provided with both was used for help.

3.1 Poker

In this problem I first developed a simple non-distributed application. I then developed a distributed application and a web service application based upon the existing application. I tried to change as little as possible from the original application. This is to illustrate the transition from a non-distributed application to a distributed application or a web service application. The development process and implementation is similar to what would be done if the applications were to be implemented from scratch.

3.1.1 Problem description

The task is to develop a program for comparing hands of poker and a client program to communicate with a user.

In a poker game to or more players have five cards each. I have put a limit of players to no more than six. A standard deck of 52 playing cards is used, with four suits and 13 ranks, and no joker. Card ranks are ordered as follows: 2, 3, 4, 5, 6, 7, 8, 9, Ten, Jack, Queen, King, and Ace. Different combinations of cards have different values.

3.1.2 Solution description

Distributed

The application adopts a client/server architecture using Java RMI.
`IPoker.java` - The poker remote interface,
`Poker.java` - The remote object implementation,
`PokerException.java` - Exception class,
`PokerServer.java` - The poker server,
`PokerClient.java` - The poker client,
`policy` - The policy file granting usage permission to the poker game.
See figure 3.1 for an illustration of the classes.

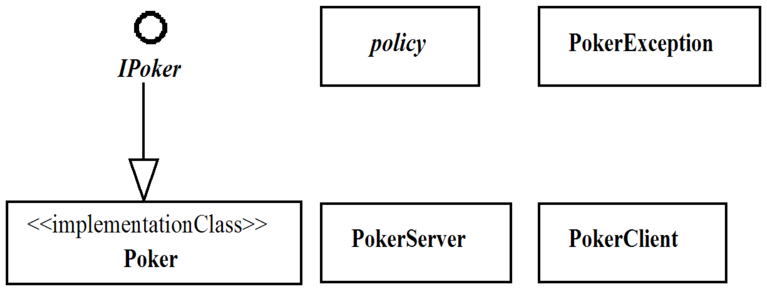


Figure 3.1: The code files

Interface `IPoker` describes the poker remote interface, see figure 3.2. It describes what the public can access, and contains only one method, `winnerHand`.

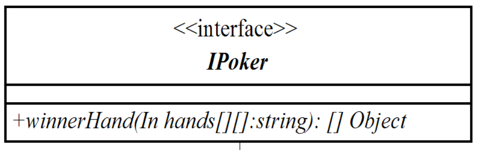


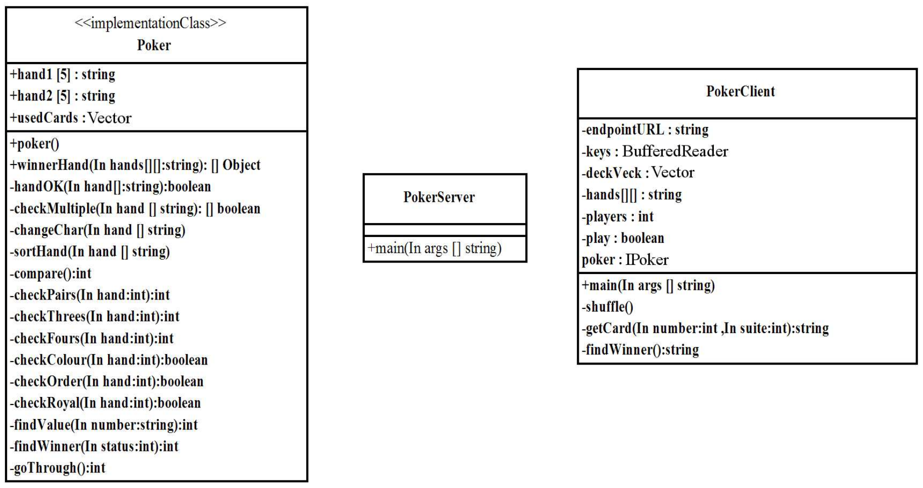
Figure 3.2: The IPoker interface

Implementation `Poker` is a remote object implementation that implements the IPoker interface, see figure 3.3.Here the abstract method is made specific. The `Poker` class also contains other methods for helping in the evaluation of which of the submitted hands is the winning hand. I decided that if an error occurred the server should throw a `RemoteException` instead of just printing the error as in the original application. This was in order for the client to see the error in case the error was due to incorrect card values.

A skeleton class, `Poker_Skel.class`, and stub class, `Poker_Stub.class` were automatically created when compiling `Poker` with the Java RMI stub compiler, *rmic*. The skeleton class is "a JRMP protocol server-side entity that has a method that dispatches calls to the actual remote object implementation." [Sun04c] The stub class is "client-side proxy for a remote object which is responsible for communicating method invocations on remote objects to the server where the actual remote object implementation resides." [Sun04c] The `Poker` class extends `UnicastRemoteObject` which is used for exporting remote objects with Java Remote Method Protocol (JRMP) and obtaining the stub.

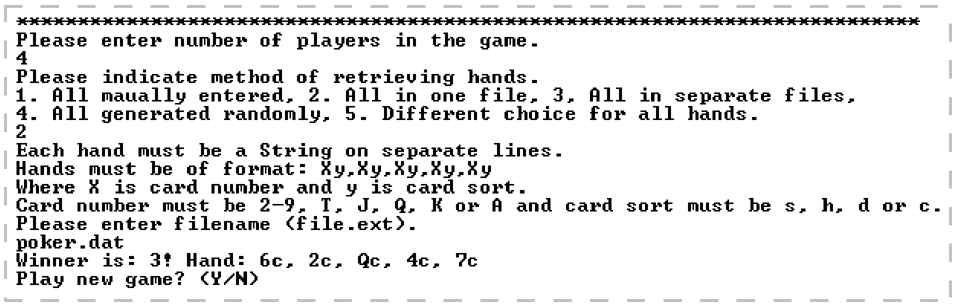
The hands are to be submitted as a double String array containing one hand (five cards) for each player. From there the application evaluates and compares two and two hands, these hands are stored as separate String arrays containing the five cards.

The methods in `Poker` are relatively straight forward and self explaining. There are methods for evaluating a given hand to find its score and for comparing two previously evaluated hands.


<<implementationClass>>
Poker
+hand1 [5] : string
+hand2 [5] : string
+usedCards : Vector
+poker()
+winnerHand(In hands[][]:string): [] Object
-handOK(In hand[]:string):boolean
-checkMultiple(In hand [] string): [] boolean
-changeChar(In hand [] string)
-sortHand(In hand [] string)
-compare():int
-checkPairs(In hand:int):int
-checkThrees(In hand:int):int
-checkFours(In hand:int):int
-checkColour(In hand:int):boolean
-checkOrder(In hand:int):boolean
-checkRoyal(In hand:int):boolean
-findValue(In number:string):int
-findWinner(In status:int):int
-goThrough():int
PokerServer
+main(In args [] string)
PokerClient
-endpointURL : string
-keys : BufferedReader
-deckVeck : Vector
-hands[][] : string
-players : int
-play : boolean
poker : IPoker
+main(In args [] string)
-shuffle()
-getCard(In number:int ,In suite:int):string
-findWinner():string
Figure 3.3: The Poker implementation, server and client

Server `PokerServer` makes an `IPoker` instance of the `Poker` and runs it, see figure 3.3. The `Naming.rebind` method binds a name to the references of the `IPoker` objects. Clients can look up the object by the reference name and remotely invoke the `winnerHand` method on the object.

Client `PokerClient` has the user interface and keeps the contact with the server, see figure 3.3. I chose to make a console based program, since a graphical user interface is not relevant to the functionality of the poker application. See figure 3.4 for an example of the client running.



```
**************************************************************************
Please enter number of players in the game.
4
Please indicate method of retrieving hands.
1. All maually entered, 2. All in one file, 3, All in separate files,
4. All generated randomly, 5. Different choice for all hands.
2
Each hand must be a String on separate lines.
Hands must be of format: Xy,Xy,Xy,Xy,Xy
Where X is card number and y is card sort.
Card number must be 2-9, T, J, Q, K or A and card sort must be s, h, d or c.
Please enter filename (file.ext).
poker.dat
Winner is: 3! Hand: 6c, 2c, Qc, 4c, 7c
Play new game? (Y/N)
```

Figure 3.4: Poker client running

The `Naming.lookup` method returns a reference, the stub, for the object associated with the name. This reference is then used when invoking the `winnerHand` method of `Poker`.

Compiling and running When the files were compiled, the `Poker` was compiled once more using the Java RMI stub compiler, *rmic*, to generate the stub and skeleton classes.

Before starting the server, the Java remote object registry, *rmiregistry*, must be started. It provides the methods used for storing and retrieving the remote object references when rebind and lookup is used.

When starting the server the servers codebase and the policy location must be specified. The server' codebase allows the stub class to be dynamically downloaded to the registry and subsequently to the client. See figure 3.5 for an illustration. The server' codebase property is set to the location of the implementation stubs. To start the server, open a console and type:

```
java -Djava.rmi.server.codebase=<codebase location>
-Djava.security.policy=<policy location>\policy
poker.PokerServer
```

The codebase location is typically `http://myhost/~myusrname/poker/`.

In the same way as for the server, when starting the client the server' codebase and the policy location must be specified. To start the client, open

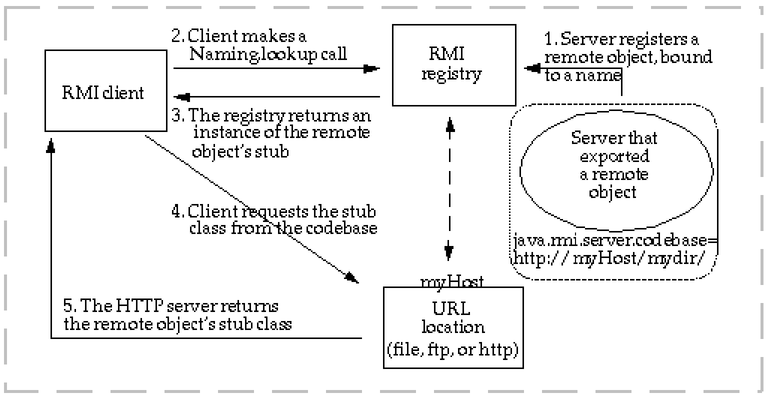

2. Client makes a Naming.lookup call
RMI client
RMI registry
1. Server registers a remote object, bound to a name
3. The registry returns an instance of the remote object's stub
Server that exported a remote object
4. Client requests the stub class from the codebase
java.rmi.server.codebase=http://myHost/mydir/
myHost
5. The HTTP server returns the remote object's stub class
URL location (file, ftp, or http)

Figure 3.5: Downloading RMI stubs. [Sun04a]

a console and type:

```
java -Djava.rmi.server.codebase=<codebase location>
-Djava.security.policy=<policy location>\policy
poker.PokerClient
```

Web Service

This application is a web service version of the distributed application poker. Even if the implementation of this application is completely independent to the implementation of the distributed application, much of the implementation is the same since they are based upon the same application. I will not describe the similarities in the implementation.
`Poker.java` - The main file,
`PokerClient.java` - The poker client,
`PokerException.java` - Exception class,
`PokerDeploy.wsdd` - Deployment descriptor,
`PokerUndeploy.wsdd` - Undeployment descriptor,
`poker.wsdl` - WSDL of the poker web service.
See figure 3.6 for an illustration of the classes

Implementation `Poker` is the implementation file, or the file to be deployed as a web service, see figure 3.6. It is very similar to the implementation in the distributed version. The major difference is that this is basically the original application file. Because the file is to be distributed with a deployment descriptor, see paragraph *Compiling, deploying and running*, nothing needed to be changed. I decided that if an error occurred an exception should be thrown (`PokerException`) instead of just printing the error as in the original application. This was in order for the client to see the error in case the error was due to incorrect cards in the hand.

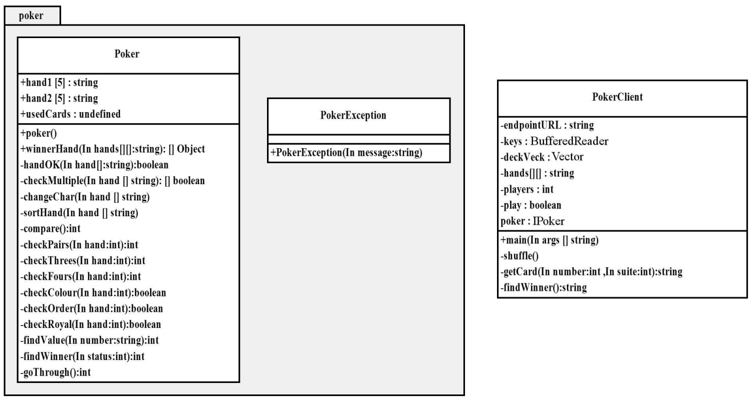

poker
Poker
+hand1 [5] : string
+hand2 [5] : string
+usedCards : undefined
+poker()
+winnerHand(In hands[][]:string): [] Object
-handOK(In hand[]:string):boolean
-checkMultiple(In hand [] string): [] boolean
-changeChar(In hand [] string)
-sortHand(In hand [] string)
-compare():int
-checkPairs(In hand:int):int
-checkThrees(In hand:int):int
-checkFours(In hand:int):int
-checkColour(In hand:int):boolean
-checkOrder(In hand:int):boolean
-checkRoyal(In hand:int):boolean
-findValue(In number:string):int
-findWinner(In status:int):int
-goThrough():int
PokerException
+PokerException(In message:string)
PokerClient
-endpointURL : string
-keys : BufferedReader
-deckVeck : Vector
-hands[][] : string
-players : int
-play : boolean
poker : IPoker
+main(In args [] string)
-shuffle()
-getCard(In number:int ,In suite:int):string
-findWinner():string

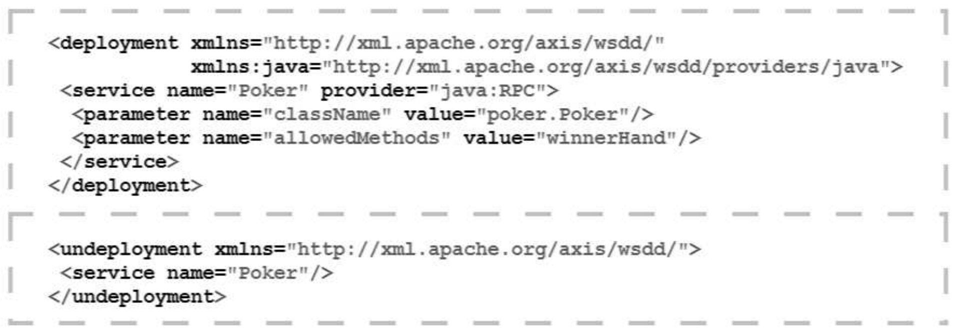
Figure 3.6: The code files

Client The web service `PokerClient` is very similar to the distributed `PokerClient`, see figure 3.6. The only difference is the communication with `Poker`. In the distributed version this is done by communication with `PokerServer`, but in the web service version the communication is done via the Apache Tomcat server [Apaa] by use of the Apache Axis technology [Apab]. The client creates new Service and Call objects. These are standard JAX-RPC (Java API for XML-based RPC) objects and are used for storing metadata about the service to invoke. The JAX-RPC can be used to build web applications and web services, by incorporating the functionality of XML-based RPC according to the SOAP 1.1 specification. The location of the service to be invoked is set by the Call objects `setTargetEndpointAddress` method. When invoking the service to utilize the deployed method in `Poker`, information about the method is set before invoking it. See figure 3.7 for code.

```java
// Send hands to the service to receive winning hand
Service  service = new Service();
Call pokerCall = (Call) service.createCall();
pokerCall.setTargetEndpointAddress( new java.net.URL(endpointURL) );
pokerCall.setOperationName( new QName("Poker", "winnerHand") );
pokerCall.addParameter("hands", XMLType.XSD_ANY, ParameterMode.IN);
pokerCall.setReturnType( org.apache.axis.encoding.XMLType.XSD_ANY );

Object[] ret0 = (Object[]) pokerCall.invoke( new Object[] { hands } );

pokerCall.removeAllParameters();
```

Figure 3.7: Create Service and Call objects and Invoking the service

Compiling, deploying and running When the files were compiled and moved to the appropriate directory (when using Tomcat it is Tomcat directory\webapps\axis\WEB-INF\classes), the web service needed to be made available for use or *deployed*. When using Axis in Tomcat this is done by using a deployment descriptor while Tomcat is running. The descriptor is written in Axis Web Service Deployment Descriptor (WSDD) format. It contains relevant information about what to be made available to the Axis engine. There should also be a file for removing the availability of the service, or *undeployment*. I used the files `PokerDeploy.wsdd` and `PokerUndeploy.wsdd`, see figure 3.8.



```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
            xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
 <service name="Poker" provider="java:RPC">
  <parameter name="className" value="poker.Poker"/>
  <parameter name="allowedMethods" value="winnerHand"/>
 </service>
</deployment>
```

```
<undeployment xmlns="http://xml.apache.org/axis/wsdd/">
 <service name="Poker"/>
</undeployment>
```

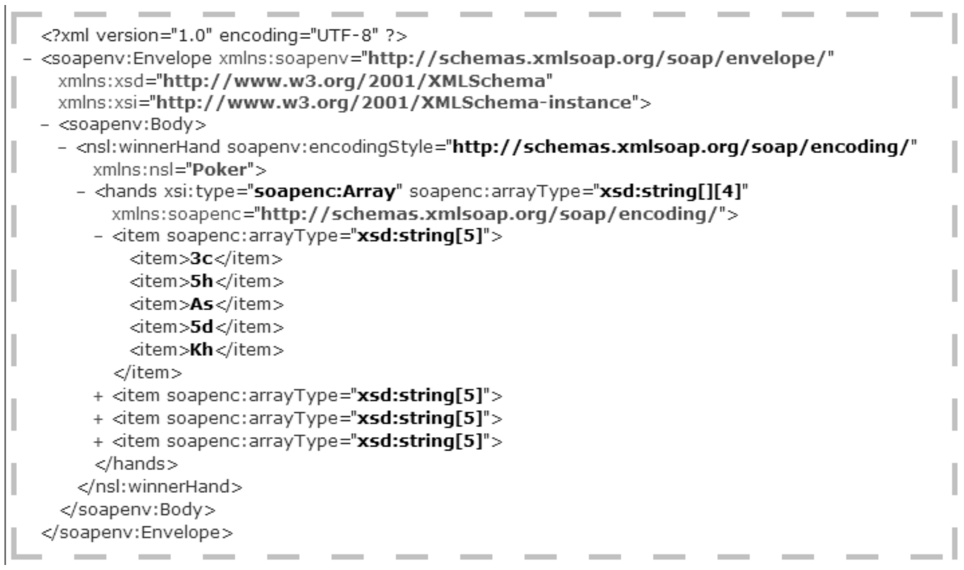
Figure 3.8: Deploy and undeploy files

The client can be run from any location. The location of the service is to be given as an argument in URL format when starting `PokerClient`. If the server where the poker web service is located is not running, the client will not be able to connect to it.

When the client's Call object try to invoke the deployed method in the poker web service, a SOAP request is sent to the server. See figure 3.9 for the SOAP request generated by the code in figure 3.7. I was regrettably not able to monitor the SOAP response from the server.

3.2 Bulletin board

In this problem I first developed a distributed application without any regard to how it could be implemented in web services. Then I developed the same application using web service technology. I tried to change as little as possible from the original application. This is to illustrate the transition from a distributed application to web service application and the challenges or problems in doing so.



```
<?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>
    - <nsl:winnerHand soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        xmlns:nsl="Poker">
      - <hands xsi:type="soapenc:Array" soapenc:arrayType="xsd:string[][4]"
          xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
        - <item soapenc:arrayType="xsd:string[5]">
            <item>3c</item>
            <item>5h</item>
            <item>As</item>
            <item>5d</item>
            <item>Kh</item>
          </item>
        + <item soapenc:arrayType="xsd:string[5]">
        + <item soapenc:arrayType="xsd:string[5]">
        + <item soapenc:arrayType="xsd:string[5]">
        </hands>
      </nsl:winnerHand>
    </soapenv:Body>
  </soapenv:Envelope>
```

Figure 3.9: SOAP request

3.2.1 Problem description

The task is to develop a bulletin board or newsgroup. The bulletin board will maintain groups or threads of messages and users may post and read messages and replies.

The bulletin board is to adopt client/server architecture. Java interfaces to define the overall design of the server are to be used with an implementation class and a server class. There is also a need of a client class to communicate with the server and to implement the user interface.

3.2.2 Solution description

Distributed

The application adopts a client/server architecture using Java RMI.
`IMessage.java` - The message remote interface,
`IBulletinBoard.java` - The bulletin board remote interface,
`BulletinBoard.java` - The remote object implementation,
`BulletinBoardServer.java` - The bulletin board server,
`BulletinBoardClient.java` - The bulletin board client,
`policy` - The policy file granting usage permission to the bulletin board.
See figure 3.10 for an illustration of the classes

Interfaces `IMessage` describes the message remote interface. It describes the design of the messages, the `set` and `get` methods. `IBulletinBoard` is the interface of the bulletin board itself. It gives an interface for keeping

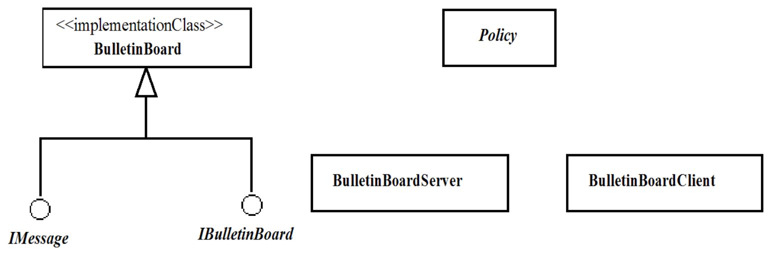
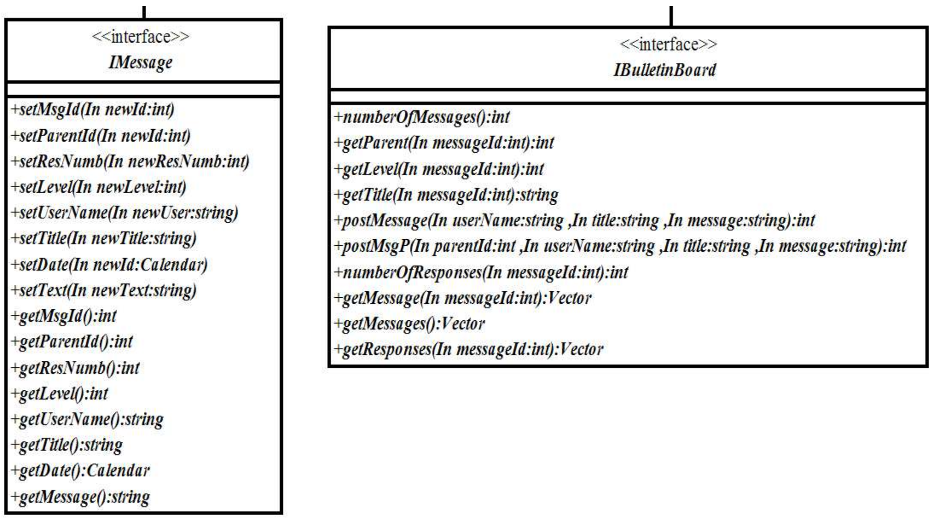


Figure 3.10: The code files

track of the messages in the bulletin board and how these are related to each other. See figure 3.11 for an illustration of the interfaces.



<<interface>>
IMessage
+setMsgId(In newId:int)
+setParentId(In newId:int)
+setResNumb(In newResNumb:int)
+setLevel(In newLevel:int)
+setUserName(In newUser:string)
+setTitle(In newTitle:string)
+setDate(In newId:Calendar)
+setText(In newText:string)
+getMsgId():int
+getParentId():int
+getResNumb():int
+getLevel():int
+getUserName():string
+getTitle():string
+getDate():Calendar
+getMessage():string
<<interface>>
IBulletinBoard
+numberOfMessages():int
+getParent(In messageId:int):int
+getLevel(In messageId:int):int
+getTitle(In messageId:int):string
+postMessage(In userName:string ,In title:string ,In message:string):int
+postMsgP(In parentId:int ,In userName:string ,In title:string ,In message:string):int
+numberOfResponses(In messageId:int):int
+getMessage(In messageId:int):Vector
+getMessages():Vector
+getResponses(In messageId:int):Vector

Figure 3.11: The interfaces

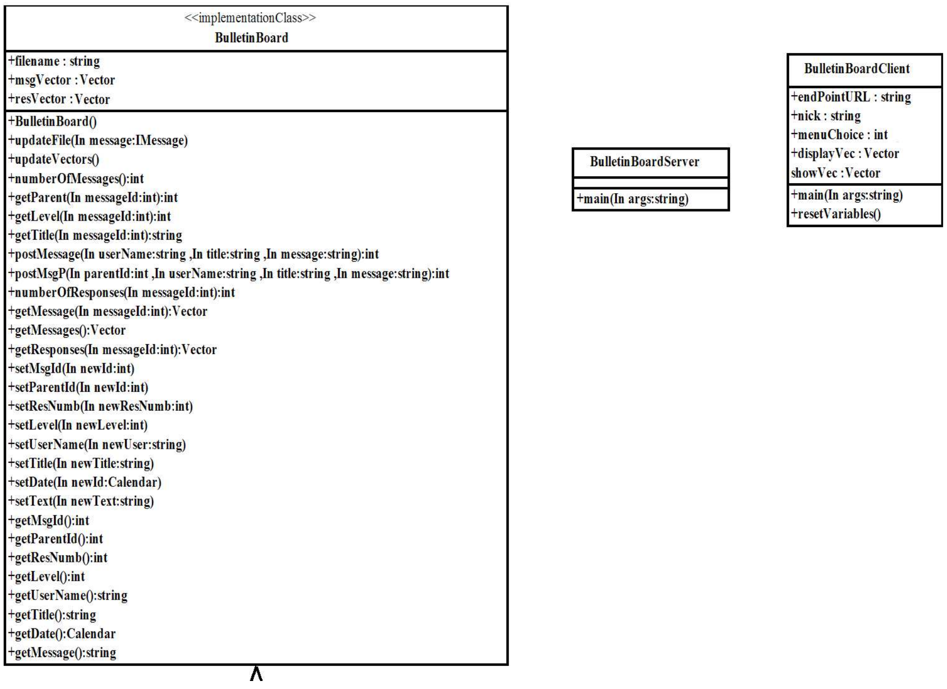
Implementation `BulletinBoard.java` is a remote object implementation that implements both interfaces, see figure 3.12. Here the abstract methods in the two interfaces are made specific.

As with the Poker application, section 3.1.2, a skeleton class, `BulletinBoard_Skel.class`, and stub class, `BulletinBoard_Stub.class` were automatically created when compiling `BulletinBoard` with the Java RMI stub compiler, *rmic*.

To keep track of the messages added to the bulletin board, I chose to use vectors. Their greatest advantages are, in my view, that they can contain mixed objects and their size can be expanded during runtime. When a new message is submitted and stored in the vector, it is also stored in a `RandomAccessFile` for later use even if the server is stopped. I chose

RandomAccessFile for storage since it is easy to add new elements at the end of an existing file. The vectors can then be updated by retrieving all the messages in the file and adding them to the vectors.

The methods in BulletinBoard are relatively straight forward and self explaining. There are IMessage methods (set and get) and IBulletinBoard methods for posting messages and retrieving messages and information relative to the messages. I chose not to send the message itself because the client may not be aware of the structure of the BulletinBoard class, instead vectors are used for transporting the message objects, where one object in the vector corresponds to one information item in the message.

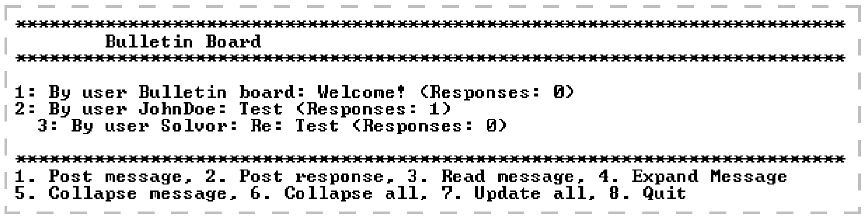


<<implementationClass>>
BulletinBoard
+filename : string
+msgVector : Vector
+resVector : Vector
+BulletinBoard()
+updateFile(In message:IMessage)
+updateVectors()
+numberOfMessages():int
+getParent(In messageId:int):int
+getLevel(In messageId:int):int
+getTitle(In messageId:int):string
+postMessage(In userName:string ,In title:string ,In message:string):int
+postMsgP(In parentId:int ,In userName:string ,In title:string ,In message:string):int
+numberOfResponses(In messageId:int):int
+getMessage(In messageId:int):Vector
+getMessages():Vector
+getResponses(In messageId:int):Vector
+setMsgId(In newId:int)
+setParentId(In newId:int)
+setResNumb(In newResNumb:int)
+setLevel(In newLevel:int)
+setUserName(In newUser:string)
+setTitle(In newTitle:string)
+setDate(In newId:Calendar)
+setText(In newText:string)
+getMsgId():int
+getParentId():int
+getResNumb():int
+getLevel():int
+getUserName():string
+getTitle():string
+getDate():Calendar
+getMessage():string
BulletinBoardServer
+main(In args:string)
BulletinBoardClient
+endPointURL : string
+nick : string
+menuChoice : int
+displayVec : Vector
showVec : Vector
+main(In args:string)
+resetVariables()

Figure 3.12: The Bulletin Board implementation, server and client

Server The structure of the BulletinBoardServer is similar to the structure of the PokerServer in section 3.1.2. The server makes a bulletin board and a message instance of BulletinBoard and runs these. The Naming.rebind methods bind names to the references of the IMessage and IBulletinBoard objects. Clients can look up these objects by the reference names and remotely invoke methods on the objects. See figure 3.12.

Client BulletinBoardClient has the user interface and keeps contact with the server, see figure 3.12. The communication with the server is similar to the communication described in the *Client* paragraph in section 3.1.2. I chose to make a console based program, since a graphical user interface is not relevant to the functionality of the bulletin board. See figure 3.13 for an example of the client running.



```
***************************************************************************
          Bulletin Board
***************************************************************************

1: By user Bulletin board: Welcome! (Responses: 0)
2: By user JohnDoe: Test (Responses: 1)
  3: By user Solvor: Re: Test (Responses: 0)

***************************************************************************
1. Post message, 2. Post response, 3. Read message, 4. Expand Message
5. Collapse message, 6. Collapse all, 7. Update all, 8. Quit
```

Figure 3.13: Bulletin board example

Compiling and running When the files were compiled, the `BulletinBoard` was compiled once more using the Java RMI stub compiler, *rmic*, to generate the stub and skeleton classes.

Before starting the server, the Java remote object registry, *rmiregistry*, must be started. And when starting the server the server′ codebase and the policy location must be specified. See section the *Compiling and running* paragraph in section 3.1.2. The server′ codebase property is set to the location of the implementation stubs. To start the server, open a console and type:

```
java -Djava.rmi.server.codebase=<codebase location>
-Djava.security.policy=<policy location>\policy
bulletin.BulletinBoardServer
```

In the same way as for the server, when starting the client the server′ codebase and the policy location must be specified. To start the client, open a console and type:

```
java -Djava.rmi.server.codebase=<codebase location>
-Djava.security.policy=<policy location>\policy
bulletin.BulletinBoardClient
```

Web Service

The application is a web service version of the distributed application bulletin board. As little as possible is changed from the original application. I will not describe these similarities.
`BulletinException.java` - Exception class
`BulletinBoard.java` - The object implementation,
`BulletinClient.java` - The bulletin board client,
`IBulletinBoard.java` - The bulletin board interface,
`IMessage.java` - Message object interface,

`BulletinDeploy.wsdd` - Deployment descriptor,
`BulletinUndeploy.wsdd` - Undeployment descriptor,
`bulletinBoard.wsdl` - WSDL of the bulletin board web service.
See figure 3.14 for an illustration of the classes

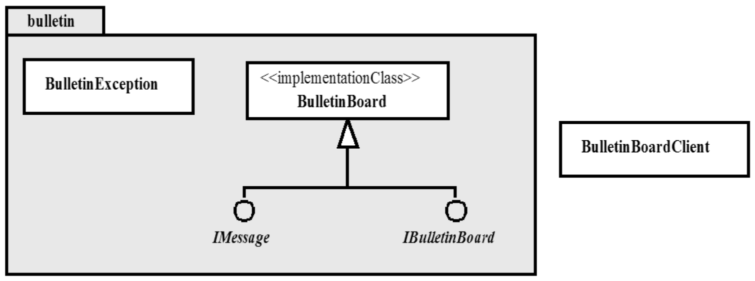


Figure 3.14: The code files

Interfaces These are very similar to the interface classes in the distributed version. `IMessage` describes the message interface. `IBulletinBoard` describes the interface of the bulletin board itself. See figure 3.15 for an illustration of the interfaces.

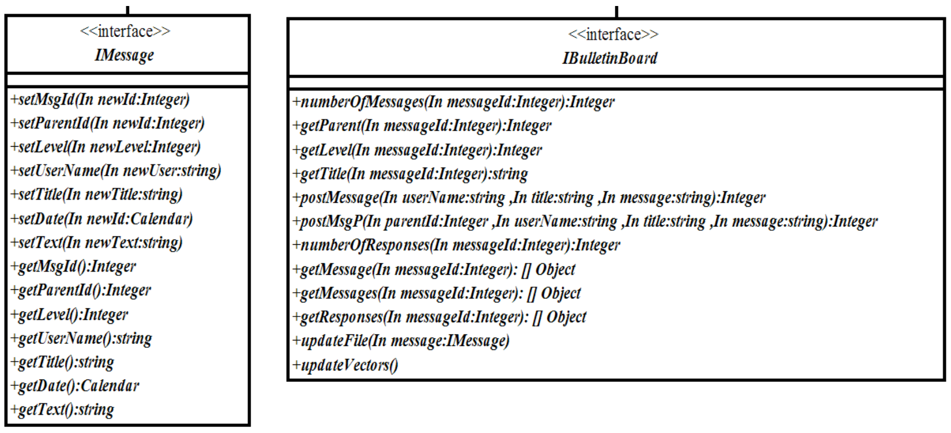

<<interface>>
IMessage
+setMsgId(In newId:Integer)
+setParentId(In newId:Integer)
+setLevel(In newLevel:Integer)
+setUserName(In newUser:string)
+setTitle(In newTitle:string)
+setDate(In newId:Calendar)
+setText(In newText:string)
+getMsgId():Integer
+getParentId():Integer
+getLevel():Integer
+getUserName():string
+getTitle():string
+getDate():Calendar
+getText():string
<<interface>>
IBulletinBoard
+numberOfMessages(In messageId:Integer):Integer
+getParent(In messageId:Integer):Integer
+getLevel(In messageId:Integer):Integer
+getTitle(In messageId:Integer):string
+postMessage(In userName:string ,In title:string ,In message:string):Integer
+postMsgP(In parentId:Integer ,In userName:string ,In title:string ,In message:string):Integer
+numberOfResponses(In messageId:Integer):Integer
+getMessage(In messageId:Integer): [] Object
+getMessages(In messageId:Integer): [] Object
+getResponses(In messageId:Integer): [] Object
+updateFile(In message:IMessage)
+updateVectors()

Figure 3.15: The interfaces

Implementation `BulletinBoard` is the implementation of the two interfaces, see figure 3.16. It is also very similar to the implementation in the distributed version, `BulletinBoard.java`. The major difference is the transportation of the message objects.

When transporting the message objects or other objects consisting of several items, I used vectors in the distributed application. But this was not adopted easily in the web service application. I therefore decided to use object arrays in these cases in the web service application. Numbers for identifying some information about a message, like the message id, are transported as the primitive type `int` in the distributed application. In the web service application I used `integers` since they are objects that can be transported in a SOAP message.

As in the distributed version I used vectors to keep track of the messages and stored them in a `RandomAccessFile` for later use. The methods in `BulletinBoard` are relatively straight forward and self explaining. They are the same as in the distributed version although parts of the method bodies are changed due to the fact that primitive types like `int` and vectors are not used for transporting information in the SOAP messages.

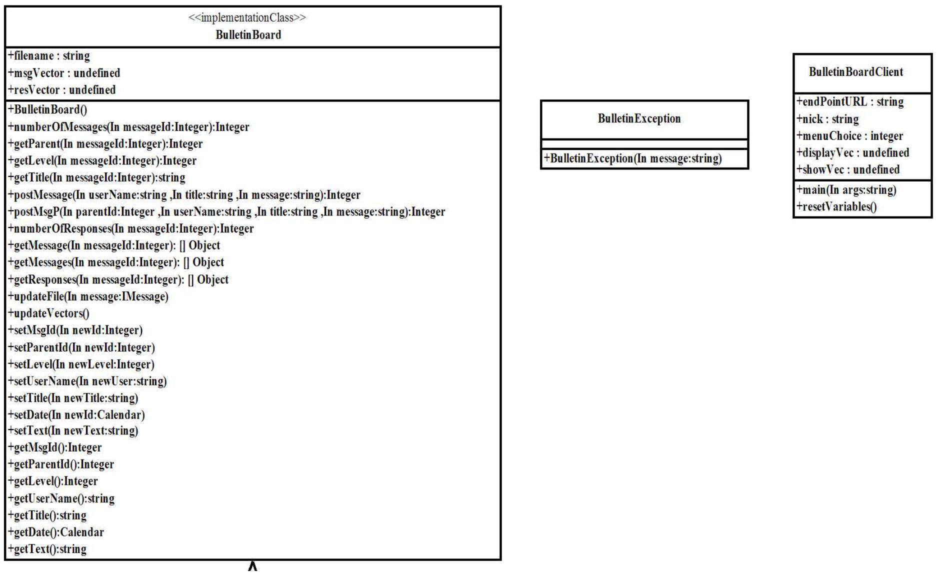

<<implementationClass>>
BulletinBoard
+filename : string
+msgVector : undefined
+resVector : undefined
+BulletinBoard()
+numberOfMessages(In messageId:Integer):Integer
+getParent(In messageId:Integer):Integer
+getLevel(In messageId:Integer):Integer
+getTitle(In messageId:Integer):string
+postMessage(In userName:string ,In title:string ,In message:string):Integer
+postMsgP(In parentId:Integer ,In userName:string ,In title:string ,In message:string):Integer
+numberOfResponses(In messageId:Integer):Integer
+getMessage(In messageId:Integer): [] Object
+getMessages(In messageId:Integer): [] Object
+getResponses(In messageId:Integer): [] Object
+updateFile(In message:IMessage)
+updateVectors()
+setMsgId(In newId:Integer)
+setParentId(In newId:Integer)
+setLevel(In newLevel:Integer)
+setUserName(In newUser:string)
+setTitle(In newTitle:string)
+setDate(In newId:Calendar)
+setText(In newText:string)
+getMsgId():Integer
+getParentId():Integer
+getLevel():Integer
+getUserName():string
+getTitle():string
+getDate():Calendar
+getText():string
BulletinException
+BulletinException(In message:string)
BulletinBoardClient
+endPointURL : string
+nick : string
+menuChoice : integer
+displayVec : undefined
+showVec : undefined
+main(In args:string)
+resetVariables()

Figure 3.16: The Bulletin Board implementation, exception and client

Client The web service `BulletinBoardClient` is very similar to the distributed `BulletinBoardClient`, see figure 3.16. The user interface is the same and the structure is the same, but the communication with the bulletin board is different. As in the Poker web service, the client creates new Service and Call objects. And when invoking the service to utilize the methods of the `BulletinBoard`, information about the method is set before invoking it. See figure 3.11 for code. There are also some other differences in the code. In the web service application, arrays are used for transporting objects consisting of several items, while vectors were used in the distributed application.

```
Service bulletinService = new Service();
Call bulletinCall = (Call) bulletinService.createCall();
bulletinCall.setTargetEndpointAddress(new java.net.URL(endpointURL));
```

```
// Posting message
bulletinCall.setOperationName(new QName("BulletinBoard", "postMessage"));
bulletinCall.addParameter("nick", XMLType.XSD_STRING, ParameterMode.IN);
bulletinCall.addParameter("title", XMLType.XSD_STRING, ParameterMode.IN);
bulletinCall.addParameter("text", XMLType.XSD_STRING, ParameterMode.IN);
bulletinCall.setReturnType(org.apache.axis.encoding.XMLType.XSD_INT );

helpInt = (Integer) bulletinCall.invoke(new Object[] {nick, title, text});

// Resetting the the parameters of the call element
bulletinCall.removeAllParameters();
```

Figure 3.17: Create service and call objects and invoking a service

Compiling, deploying and running Just as in the Poker web service, when the files were compiled and moved to the appropriate directory, the web service needed to be made available for use or *deployed*. See section 3.1.2, *Compiling, deploying and running*. I used the files `BulletinDeploy.wsdd` and `BulletinUndeploy.wsdd` for deployment and undeployment.

As with the Poker web service, the client can be run from any location if the service location is given. And a SOAP request is sent to the server, when the client's Call object tries to invoke some method in the bulletin board service. See figure 3.18 for the SOAP request generated by the code in figure 3.17.

```
<?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>
    - <nsl:postMessage soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        xmlns:nsl="BulletinBoard">
        <nick xsi:type="xsd:string">Test</nick>
        <title xsi:type="xsd:string">Test message</title>
        <text xsi:type="xsd:string">This is a little test message :)</text>
      </nsl:postMessage>
    </soapenv:Body>
</soapenv:Envelope>
```

Figure 3.18: SOAP request

Chapter 4

Results and Discussion

In this chapter I will look at the solutions from chapter 3, especially the web service applications. The distributed applications were based on earlier knowledge on how to develop distributed applications using Java RMI, but the web service solutions are my first attempt to develop web service applications. I had previously only examined web services at a theoretical level. The challenges and problems I encountered may be typical for first time developers of web services.

4.1 Implementation solutions

I chose the user interface to be console based in both the distributed and the web service applications. A graphical user interface (GUI) is not relevant to the functionality of the applications. On the other hand it is possible to make applications with GUIs or applets to be used on web for all the applications.

4.1.1 Poker

Both poker applications are based on a non-distributed application with a main file (`Poker`) and a client (`PokerClient`). The source code for these is listed on the enclosed CD, the source code for the original, the distributed and the web service applications are all listed on the CD. See figure 4.1 for an illustration of the connection between the distributed and the web service version of `Poker`. The files which are distinctive for the distributed application are in the grey area, the files which are distinctive for the web service application are in the white area, and the files which are common are in the grey and white shaded area.

The main workload in the development is on the client side. On the server side, the `Poker` class just evaluates the best of the submitted hands. The game information is to be handled by the client.

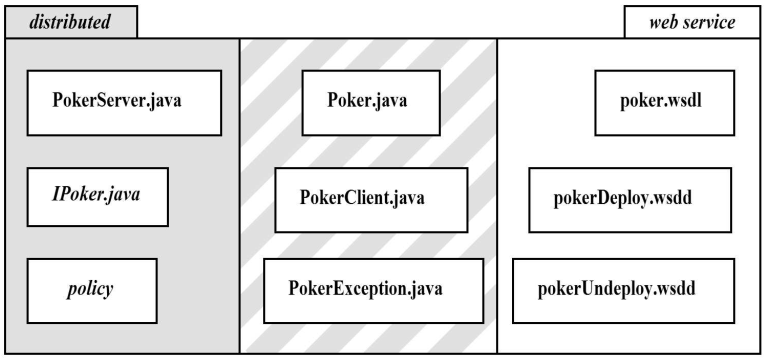


Figure 4.1: Poker: distributed and web service

The development of the applications was not very difficult. Only small changes from the original non-distributed application were necessary on both the server and client side to make the distributed and the web service applications.

4.1.2 Bulletin board

The web service bulletin board application is based on the distributed bulletin board application. The source codes for both applications are on the enclosed CD. See figure 4.2 for an illustration of the connection between the distributed and the web service version of `Bulletin Board`. The files which are distinctive for the distributed application are in the grey area, the files which are distinctive for the web service application are in the white area, and the files which are common are in the grey and white shaded area.

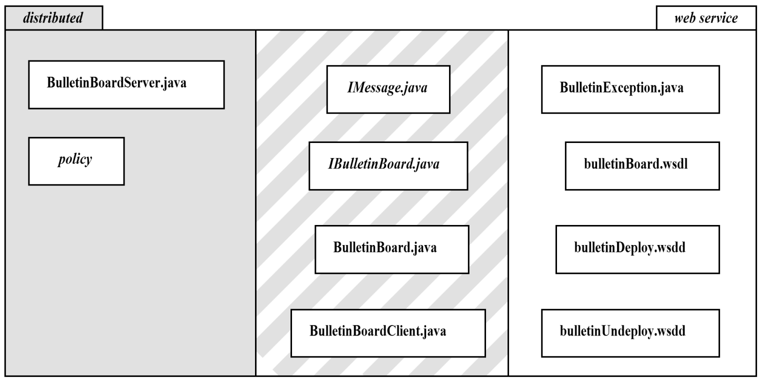


Figure 4.2: Bulletin Board: distributed and web service

The main workload in the development is on the server side. The client just sends or displays messages and communicates with the user. On the server side the messages are added, retrieved, and evaluated in various ways.

The development of the distributed application was not very difficult. I had previously made client/server Java RMI applications on the same size as the bulletin board application. In addition there is a lot easier to understand documentation of how to make such applications.

The web service application was not difficult, but I encountered a challenge on how to make sure that the variables in the `BulletinBoard` are not lost. I chose to only use the existing feature of storing and retrieving messages from a `RandomAccessFile`. However this solution may not be sufficient because data may be lost due to poor concurrency control.

4.1.3 Deployment and running

The deployment of the distributed applications is simply to run the application server. The public methods of the application are accessible to the client as long as the application server is running.

Details on how to compile and run the distributed applications are described in the *Compiling and running* sections (3.1.2 and 3.2.2).

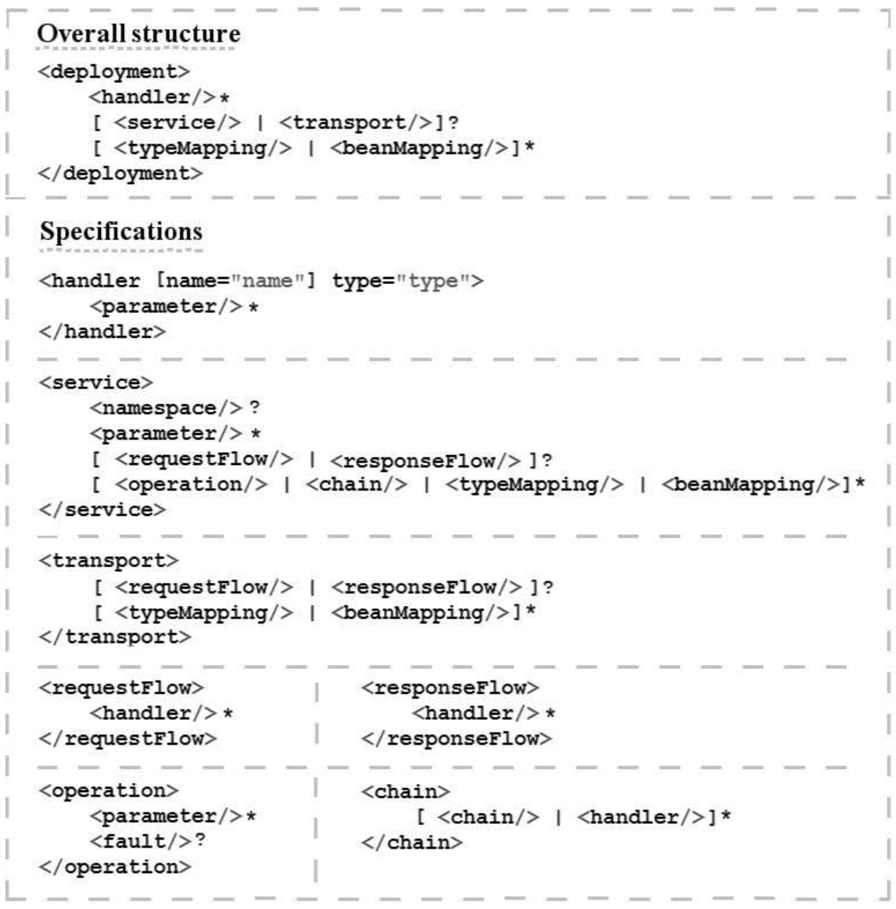
The deployment of the web service applications is to deploy them to a running server, I chose the open source server Apache Tomcat [Apaa] since it is free of charge and easy to use. The deployment was done using a deployment descriptor written in Axis Web Service Deployment Descriptor (WSDD) format, see figure 4.3 for an illustration of the structure. When deploying the poker web service, a WSDL document is automatically generated for the published web service (files `Poker.wsdl` and `BulletinBoard.wsdl`). When using other server types and other technologies, there may be other ways to deploy and undeploy a web service. See the *Compiling, deploying and running* sections (3.1.2 and 3.2.2) for more details. The public methods of a web service application are accessible to the client as long as the Tomcat server is running.

4.2 Challenges and problems

During the implementation of the web service applications I encountered some challenges and problems which may be typical for first time developers of web services.

4.2.1 Documentation

The major challenge in the development process was the lack of good and relevant documentation. There is much documentation on how to make



Overall structure

```
<deployment>
    <handler/>*
    [ <service/> | <transport/>]?
    [ <typeMapping/> | <beanMapping/>]*
</deployment>
```

Specifications

```
<handler [name="name"] type="type">
    <parameter/> *
</handler>

<service>
    <namespace/> ?
    <parameter/> *
    [ <requestFlow/> | <responseFlow/> ]?
    [ <operation/> | <chain/> | <typeMapping/> | <beanMapping/>]*
</service>

<transport>
    [ <requestFlow/> | <responseFlow/> ]?
    [ <typeMapping/> | <beanMapping/>]*
</transport>

<requestFlow>                  <responseFlow>
    <handler/> *                   <handler/> *
</requestFlow>                 </responseFlow>

<operation>                    <chain>
    <parameter/>*                  [ <chain/> | <handler/>]*
    <fault/>?                  </chain>
</operation>
```

Figure 4.3: Web Service Deployment Descriptor (WSDD)

distributed application using the programming language and development environment of your choice. But the documentation on how to make a web service is mostly for small web services like the poker application. When developing a medium or large sized web service you are basically on your own.

Since I chose the Java 2 Platform, Standard Edition (J2SE) [Sunb] in the distributed applications, I decided to do the same in the web service applications to better highlight the differences. Most of the documentation I could find for web services based on Java J2SE was either for small web services or unsuitable for my web service applications. This is due to the desired comparison to the distributed applications. Parts of the web service code may therefore be influenced by the fact that the solution worked well in the distributed application and there was some trial and error in the beginning of the development of the web services.

The poker web service application is similar in size and implementation as many of the examples on how to make web services using Java J2SE. Most other Java examples were on how to make web services using Enterprise JavaBeans (EJB) or Java Servlets from Java 2 Platform, Enterprise Edition (J2EE) [Suna], or Java Web Services [Sunc]. These technologies were not as suitable for the comparison with the distributed applications as the Java J2SE.

If I had chosen the programming language and environment for the web service applications independently of the distributed applications, the documentation may have been better and more relevant to the application solution. Other programming languages and environments, like the Microsoft .NET platform, may have better documentation on how to make medium or larger sized web services than what I could find for my web service solutions. Hence the solutions to the problems may have been better or more illustrating of the web service technology.

In my opinion, the documentation provided with Apache Tomcat and Apache Axis was too thin, especially the documentation provided with Axis. The installation instructions and the API were good. But the explanation and examples of how various aspects of web services in Axis should be implemented and how they work, were not satisfactory. However, this is probably due to the fact that it is open-source and the documentation was at the time of the development partly still under construction.

4.2.2 Variable storage

Another challenge in the development of the web service applications was how to preserve variables. In the distributed application a java RMI server is running and makes sure that the variables are not lost during runtime. But in the web service application, the only running server is the Tomcat server [Apaa].

The task of the Tomcat server in this context is to pass on the incoming messages to the correct web service and the responses to the correct requester (client). It has no other connection to the web service application and will not be able to preserve the variables unless extra features have been added. The web service application itself is dormant between each time it is called and the variables are lost between each call. In the development of the poker web service this caused no problem because it only evaluates the incoming hands and responds with an appropriate value. But in the bulletin board web service, the messages must be kept for later use.

My solution to the problem was to use the existing feature of storing in and retrieving messages from a `RandomAccessFile`. I could also have chosen other types of storage, for example a database. Another solution could have been to implement `BulletinBoard` as a running application which could preserve the messages, and to add additional features to the web service to make sure that Tomcat sends the incoming messages to the running `BulletinBoard`. However this is only a theory, I have not explored how this would be done in practice.

4.3 Alternative Solutions

There are possibly several alternative solutions to the problems in chapter 3. The most obvious alternative is to use another programming language or environment. Some of them may be better suited for developing the web service applications, especially the bulletin board web service.

4.3.1 Development process

The description of problems to be implemented were quite vague and did not have many clear requirements. If the problems had been specified in more detail, the solutions could have been different.

During the implementation process, I first designed the class diagrams displayed in chapter 3. Then I started to develop the programs based on these diagrams and the problem description. If the given problems had been larger, the problem descriptions better, there had been others involved in the development process, or the designated time for the development process longer, the planning would have been better. The planning would then probably have a longer and more thorough problem analysis and implementation design process. Such a development process would probably be based on a software engineering process like the Rational Unified Process, see the book by Philippe Kruchten [Kru04] for more information.

4.3.2 Poker

There are lots of examples, on the web and elsewhere, of small and simple web services on the same size as the poker web service using various programming languages and environments. The structure of many of them could probably be a good alternative to the poker web service.

Other alternatives to the poker web service could be how the hands submitted to `Poker` are evaluated to find the winner. My solution is based on information found when searching on `www.google.com` about poker games and the rank of different combinations of cards.

The `PokerClient` could also be structured differently or have other options. It contains few methods where the `main` method is a large method. This method has a large switch-case part which may be partly split up and much of the functionality put in separate methods. This solution would probably be a much cleaner solution than the existing one. The same goes for the distributed `PokerClient`.

4.3.3 Bulletin board

The examples I could find on web services on the same size as the bulletin board web service, were based upon other programming languages and platforms. I believe the development process and the implementation of the web service would be less complicated and perhaps more true to the web service technology if I had chosen another programming language or platform.

I believe that the best solution to the bulletin board problem is to have either a class or an interface defining the structure of the messages and a class or an interface defining the structure of the bulletin board itself. The alternative solutions to the problem would probably have changes in the keeping and storage of the variables and messages, the deployment of the web service, the communication between the client and the service, or in all these three areas.

For example there could be a running bulletin board server, somewhat similar to the distributed `BulletinBoardServer`, that would take care of keeping and storage of the variables and messages. How this web service should to be deployed and the communication between the client and the web service would depend on the programming language and platform chosen.

The `BulletinBoardClient` could also be structured differently. It contains only two methods where the `main` method is a large method containing a large switch-case structure. Much of the functionality of the switch-case could perhaps be put into separate methods, and the switch-case only call these methods. This solution would probably be a much cleaner solution than the existing one. The same goes for the distributed `BulletinBoardClient`.

4.3.4 Poker expansion

The poker web service can be used as a basis to other distributed or web service applications. For instance a poker game application may be the client to the poker web service and call it to evaluate the winner in a game. It will then be possible to combine distributed and web services, for instance by calling the poker web service from a distributed application. See figure 4.4 for an illustration. If I were to develop other solutions than the centralized poker and bulletin board solutions, they would probably be similar in structure to this solution.

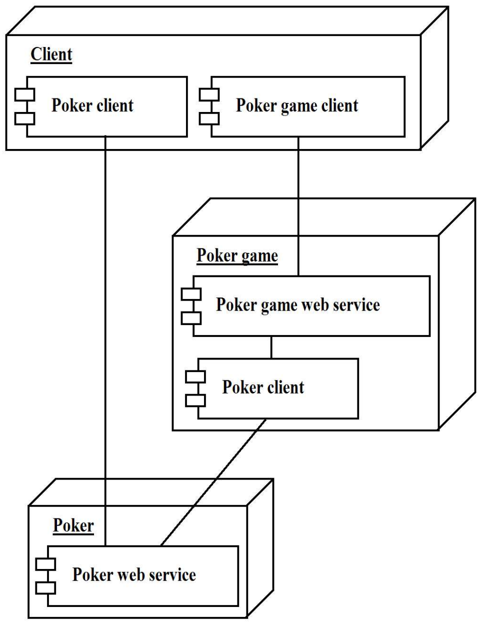


Figure 4.4: Possible use of the poker web service

This web service or the distributed application would need to store game variables during runtime. Such variables could be a deck of cards, the waste cards, the various players and the cards dealt to them. How these variables should be kept during runtime and stored depends on the programming language and platform chosen, but the challenge would be similar to the one in the bulletin board web service. However, it would not be a practical solution to store all the variables permanently. Variables like the hands will only be necessary during a round in a game, so these should not be stored. On the other hand, the winner of a round or a game could be stored in a high-score list.

The poker game web service could also include some of the features from the bulletin board web service to allow for chat between the players in the game. The challenge of how to store the variables only during runtime would also apply to this.

4.4 Conclusion

As stated in the beginning, the web service solutions in chapter 3 were my first attempts to develop web service applications. I believe the solutions are influenced by that I had previously only examined web services at a theoretical level, but had knowledge about how to develop distributed applications using Java RMI. If I had knowledge about web service applications development as well, the web service solutions and the challenges and problems I encountered would possibly have been different.

Chapter 5

Evaluation

In this chapter I will evaluate the content of previous chapters. I will look at both the theoretical, chapters 1 and 2, and the practical, chapters 3 and 4. The intention is to evaluate distributed systems and web services to see if web services can be used as foundation for distributed systems.

I will also evaluate the process of writing this thesis. Here I will try to sum up the process as well as try to comment on my own work.

5.1 Distributed systems and web services

In chapter 1 I defined distributed systems to be systems where different components in a network, communicate with each other and coordinate their actions only by passing messages. A component was defined to be a program execution on a computer or a device such as a computer or a printer. Web services are based upon the principles of distributed systems. In chapter 2 a web service was defined to be a set of functions that are published to a network for use by other programs. Like for distributed systems, the communication in web services is done only by passing messages. A web service and communicating programs may be implemented on any platform and in any programming language.

By only looking at this it may seem as if web services are only applicable to some areas of distributed computing. Many regard these areas to only involve publishing software services on the Internet via browsers, while others regard these areas to possibly apply to any area of distributed computing. In my opinion the realities and possibilities are probably somewhere in between.

When referring to distributed systems in this section, I mean distributed systems in general except web services unless otherwise stated.

5.1.1 Comparison

Web services have inherited some of the characteristics from distributed systems. Hence some characteristics are very similar for distributed systems and web services, however others are quite different.

Advantages and disadvantages

There are advantages and disadvantages in both distributed systems and web services. Some of these are the same in both while others are different from distributed systems to web services, in addition some are advantages with challenges or disadvantages. See tables 5.1 and 5.2 for a listing of benefits and challenges in distributed systems and web services, and in which sections these are described. The common benefits, the common disadvantages and challenges, and the differences in advantages and disadvantages are described in corresponding paragraphs below.

Benefit	*Applies to*	*Described in*
Discrete	WS	2.2
Easy to implement, understand and use	WS	2.2
Heterogeneity	WS	1.2
Industry support	DS & WS	2.2
Interoperable	WS	2.2
Openness	DS & WS	1.2
Resource sharing	DS & WS	1.2
Reusable	WS	2.2
Scalability	DS & WS	1.2
Transparency	DS & WS	1.2
Ubiquitous	DS & WS	2.2
Understandable	WS	2.2

Table 5.1: Benefits in distributed systems (DS) and web services (WS)

Common benefits Some of the advantages which are common for distributed systems and web services are:

Industry support: There are several leading hardware and software vendors which support distributed systems and the web service standards.

Openness: A system's ability to expand and re-implement, for example how well new resources can be added to it, defines the system's openness. It may be a clear advantage that a system has a high degree of openness. However the openness may be difficult to implement, hence it presents a challenge for both distributed systems and web services.

Resource sharing: The desire to share resources is one of the key motivations for any distributed system. What resource is to be shared and to what extent, depends on the system.

Scalability: If a system is able to remain efficient, both in performance and in resource use, after a significant increase in the number of resources and users, it is described as scalable. Such scalability is favourable although it may present a challenge to developers.

Transparency: The consequence of distribution is hidden by the system's transparency. There are many types of transparency which all are important and beneficial to the system. The various transparency types are: access, location, persistence, relocation, concurrency, replication, failure, performance, and scaling transparency. However, it may present a challenge to design and implement the transparencies.

Ubiquitous: Any device which supports the technologies used in a distributed system or web service can access it. On the other hand, if the device is to host the distributed system or web service some additional features may be required.

Challenge	*Applies to*	*Described in*
Concurrency	DS & WS	1.2
Fault tolerance	DS & WS	1.2
Heterogeneity	DS	1.2
Openness	DS & WS	1.2
Quality of service	DS & WS	2.2
Scalability	DS & WS	1.2
Security	DS & WS	1.2 & 2.2
Time	DS & WS	1.2
Transaction	DS & WS	1.2
Transparency	DS & WS	1.2

Table 5.2: Challenges in distributed systems (DS) and web services (WS)

Common disadvantages and challenges Some of the disadvantages and challenges which are common for distributed systems and web services are:

Concurrency: It is desirable to maintain resource integrity in a system. This is handled by a system's concurrency control which synchronizes concurrent access to the same resources. In the bulletin board web services the concurrency control to the `RandomAccessFile` is somewhat sufficient since the application is trivial and just an example. However, if the application had been more relevant, the concurrency control would be insufficient.

Fault tolerance: How to deal with failures represents a challenge for any system. Classes of failures are listed in table 1.1. There are various techniques for dealing with faults: detecting failures, masking failures, tolerating failures, recovery, and redundancy.

Openness: It is an advantage to have a system with a high degree of openness. But the openness may cause a challenge for the system designers, especially if the system is complex and is implemented and managed by various people.

Quality of service: To ensure the quality of transferred data and that it is processed in a specified amount of time is important in any system. But it may be a challenge to ensure the quality of service characteristics in distributed systems and in web services, especially if a web service is accessed from an external source, since the quality of the web service may be unknown.

Scalability: It is beneficial to have a scalable system, although there are challenges in designing and implementing such a system. Some of these challenges include controlling costs of resources, controlling performance loss, preventing the systems software resources of running out, and avoiding bottlenecks.

Security For most users of a system, security is very important, and it is likely that they will not use the system if they feel it is too insecure. There are several types of potential threats which are divided into three classes: leakage, tampering, and vandalism. Most of these threats are applicable to both distributed systems and web services, as are the various security techniques used for dealing with them. Web services have an additional security measure in the web service security standard (WS-Security).

Time: Algorithms and applications depend on time for coordinating events and timestamps that are used in various system tasks. Every computer has its own physical clock, and it may be a challenge to synchronize them. Fortunately there are several algorithms for approximate synchronizing of clocks.

Transactions: In section 1.2 I stated that the goal of a transaction is to control access to shared resources and that either the transaction is completed or that nothing has happened to the data. This is ensured by the transactions ACID characteristics. They are atomicity, consistency, isolation, and durability. It may be a challenge in design and implementation to ensure these characteristics.

Transparency: To hide the consequences of distribution may present a challenge in design and implementation. Fortunately not all the transparency types may be applicable to a system, and need not be implemented.

Differences There may be some characteristics in distributed systems and web services which may be an advantage in one and a disadvantage or present challenges in the other. Some of these characteristics are:

Discrete: Web services are discrete in that each web service typically only provides a single piece of functionality and is completely independent. For this reason, a web service may not be practical as a large system. Distributed systems on the other hand, may be larger systems providing multiple functionalities and may be dependent on other parts or systems.

Easy to implement, understand and use: When implementing parts of a distributed system, knowledge of the other parts of the system is necessary to communication between the various parts. Web services may be implemented without previous knowledge of the target system as long as the web service standards are respected. However I did not find it easy to implement parts of the bulletin board web service because of the challenge of preserving the variables during runtime. On the other hand, this challenge may not have arisen if I had more practice in developing web services.

Heterogeneity: To ensure that system works properly even if there are variations and differences between components, may be a challenge in distributed systems. These challenges are trivial when using web services. The communication with a web service takes place by passing messages. The structure of these messages is described in the web service' WSDL document and published via UDDI. The messages are sent via the SOAP messaging protocol. See section 2.3 for descriptions of WSDL, UDDI and SOAP.

Interoperable: A web service can interact with any other web service as long as the web service standards are used and respected. But before a distributed system or parts of a distributed system to interact with other distributed systems, a common communication channel and means must be agreed upon.

Reusable: A web service can be extended and reused whenever necessary. This is only partially true for distributed systems, as some distributed systems can only be used for their initial indented purpose.

Understandable: Web services are understandable for both humans and computers due to the characteristics of web services. Distributed systems are obviously understandable to computers, but they require additional documentation to be understandable to humans. Detailed documentation and descriptive comments in the source code are examples.

Other characteristics

There are other characteristics in distributed systems and web services than just the advantages and disadvantages. Some of these are similar for both while others are unique for distributed systems or web services. I will not describe the all characteristics, nor in detail, as this is done in the previous chapters. See table 5.3 for a listing of the characteristics of distributed systems and web services, and the sections where these are described.

Characteristic	*Applies to*	*Described in*
Data representation	DS & WS	2.3.1
Distributed file system	DS	1.3
Distributed programming	DS & WS	1.3
Internet	DS & WS	1.3
Intranet	DS & WS	1.3
Mobile devices	DS & WS	1.3
Name service	DS & WS	1.3
Networks	DS & WS	1.3
Remote communication	DS & WS	1.3
SOAP	WS	2.3.2
System patterns	DS & WS	1.3
UDDI	WS	2.3.4
WSDL	WS	2.3.3

Table 5.3: Characteristics in distributed systems (DS) and web services (WS)

Communication Both distributed systems and web services take advantage of some of the following features:

Name service: Names are used for referring to resources. When communicating with a resource its name, address and attributes are important. There are various name services which are used for retrieving the names, addresses and attribute; the Domain Name System (DNS) is probably the most known. A name service is in principle a distributed service and is used by other distributed services and web services as well.

Networks: Both distributed systems and web services use networks for communication. There is a variety of types of networks which are classified by size and usage. Se table 1.3 for comparison.

Internet: The Internet is a world-embracing set of computer networks, all linked together as one. It is one of the most typical distributed systems and is used by other distributed services and web services for communication purposes.

Intranet: The intranets are subnetworks to the Internet network which are operated by companies or organisations. They are, like the Internet, distributed systems themselves and are used by both other distributed services and web services for communication purposes.

Mobile devices: Portable devices which are able to connect to networks in different places are called mobile devices. They can be part of a distributed system. Some of these mobile devices support the technologies used in web service, and can be used to access a web service.

Remote communication: There are several technologies for remote communication which are used by both distributed systems and web services communicating and invoking operations elsewhere. Remote procedure call (RPC), remote method invocation (RMI), and event-based programming model are examples of such technologies. Hamish Taylor [IT03, Lec. 14]gives a good comparison of some of the characteristics in some of the remote computing technologies, see table 5.4.

Architecture	*Payload Format*	*Endpoint Address*	*Wire Protocol*	*Interfaces*
ONC RPC	eXtensible Data Representation	host & program number	ONC protocol (binary)	RPC specification
RMI	serializable Java datatypes	URL	JRMP (binary)	Java interfaces
CORBA	Common Data Representation	IOR	IIOP (binary)	IDL
Web Services	XML	URL	SOAP (textual)	WSDL

Table 5.4: Comparison of Distributed Computing Technologies

Data representation Data sent when communicating with or in a distributed system or with a web service should conform to predefined datatypes, message format, and structure specification. When communicating with a web service it is predefined that this should be based on the specifications of structures and datatypes in the W3C XML specification. But when communicating with or in a distributed system, the specification must be set individually for each system.

Distributed file system In a distributed file system persistent data are available across a network. A distributed file system is itself a distributed system, and there are several examples of well functioning distributed file systems. Web services may use distributed file systems, but I think it is impractical and slightly pointless to make a distributed file system by only using web service technology.

Distributed programming Distributed applications run in distributed environments and make use of the characteristics of distributed systems. There are no restrictions with respect to technology, platform or programming language. Some programming languages and architectures, like CORBA, Java, and DCOM, have built in functionalities which make the source code less complex. Web service applications can use the functionalities of such programming languages and architectures. Additionally a web service application can be described as a distributed application using web service technology.

System patterns The system pattern or topology defines how the components in the system fit together. There are some basic patterns which are widely used in distributed systems. These are the centralized, decentralized, hierarchical and ring systems, there are also various hybrids of these. Web services typically use the centralized system pattern, but can potentially use other patterns as well.

5.1.2 Conclusion

As stated in the beginning of this section, by only looking at the definitions of distributed systems and web services it may seem as if web services are only applicable to some areas of distributed computing. Web services contain many features which satisfy the goals of distributed systems. And in many cases they will probably be the appropriate way of designing a distributed system. However in my opinion there are limits to when web services can be used as foundation for distributed systems. My views are probably influenced by the fact that the web service solutions in chapter 3 were my first attempts to develop web service applications and that I had previously only examined web services at a theoretical level.

I believe that web services are best suited for smaller applications with the same size as the poker application. In larger applications and in file and database sharing, one should be able to trust the server side. Since one may not know if the server application or the publisher is reliable, web services could be too risky to be trusted in major applications. However it is important to point out that the technology itself is not risky.

In my opinion there is no single correct answer to the question of whether to use a distributed or web service solution for any given task. I believe it is better to use a distributed solution internally in an organisation or system and to use a web service solution in smaller and less trivial solutions, especially if the web service is made by an external party. However, designers and developers should analyse each problem or task individually to evaluate whether to use a distributed or web service solution. I believe that as the technology evolves and new and better ideas are developed, the interest for and the quality of web services will probably grow. This may produce better and faster web services and less trivial publicly available web services.

5.2 Writing process

Finally it is time to look at the work process and the solution approach to the thesis.

As stated in the preface, the given problem description was *web services / distributed systems. Can web services be used as foundation for distributed systems?*. It was also given that the thesis should cover: *overview of distributed systems*, *overview of web services*, *attempt to develop web services*, and *evaluation*. The chapters are more or less based on the given problem description and contents.

Looking back Early in the work process I made a rough sketch of the work process, where each progress step was outlined for each month. For each step I made a more detailed plan outlined for each week. I have tried to stick to this plan as much as possible. In the original plan the goal was to finish early in case anything unexpected occurred. This was fortunate, because the gap between the expected finish and the thesis due date was reduce due to external reasons.

The steps to the work process and their sequence were similar to the chapter division. The writing itself was not very difficult. At times the writing was slow, but that was mainly because there was too much information to be included in the thesis.

In the practical portion I encountered some challenges. These challenges were probably due to the fact that I had previously only examined web services at a theoretical level and the web service solutions were my first attempts to develop web service applications that. The practical portion is more closely described in chapter 4.

The evaluation of both the practical aspect and the whole process was not difficult, but required more profound thinking and some self-criticism. However, if the evaluations were good is not for me to decide.

Looking ahead In my opinion a continuation of the thesis and the exploration of the subject *can web services be used as foundation for distributed systems* should be to explore in more detail the practical aspect. As stated in the conclusion in 5.1.2, web services contain many features which satisfy the goals of distributed systems. However, there is no single correct answer to the question of whether to use a distributed or web service solution to any given task.

One approach could be to analyse a given distributed system to evaluate if it is possible to replace the existing system solution with a web service solution. It should also be evaluated if it is desirable to replace the existing system solution due to factors like security, efficiency, and quality of service. The system to be evaluated should be larger than the small ones

I developed in chapter 3, to better analyse further sides than what I could. The system should also be an existing system to avoid spending time on analysis and design for a distributed solution as well as the web service solution.

Another approach could be to try to develop a distributed system using web service technology. The requirement description of the system should be given to ensure that the quality of the system requirement and the system size are satisfactory.

An additional approach could be a practical thesis based upon this thesis. The thesis would not include much theory as this is already done in this thesis.

I believe that if possible the continuation should be a long thesis, regardless of the chosen approach. This thesis is a short one, and I found the time to be insufficient to explore various aspects in more detail.

Conclusion Both the subject and the writing process were interesting. I have learned a good deal from the writing process itself in addition to what I learned about distributed systems and web services, perhaps especially from minor slips and misjudgements during the process. In my opinion the subject of distributed systems and web services and whether web services can be used as a foundation for distributed systems, is an interesting subject which should be explored further.

Bibliography

[Apaa] Apache Jakarta Project. *Apache Tomcat 5.0.19*. Downloadable from jakarta.apache.org/tomcat.

[Apab] Apache Web Services Project. *Apache Axis 1.1*. Downloadable from ws.apache.org/axis.

[C$^+$01] George Coulouris et al. *Distributed Systems - Concepts and Design*. Addison-Wesley, third edition, 2001.

[Cap01] CapeClear Software Ltd. *Cape Connect Three Concepts. Understanding Web Services*, 2001. www.capeclear.com/products/manuals/three/Concepts/html.

[CC02] IBM Corporation and Microsoft Corporation. Security in a web services world: A proposed architecture and roadmap. *IBM developerWorks: Web services*, 2002. www-106.ibm.com/developerworks/webservices.

[Cro96] Jon Crowcroft. *Open Distributed Systems*. Artech House, 1996. Also published on web on www.cs.ucl.ac.uk/staff/J.Crowcroft/ods/ods.html.

[dev02a] IBM developerWorks. Web services security policy (ws-securitypolicy). *IBM developerWorks: Web services*, 2002. www-106.ibm.com/developerworks/webservices.

[dev02b] IBM developerWorks. Web services security (ws-security). *IBM developerWorks: Web services*, 2002. www-106.ibm.com/developerworks/webservices.

[dev02c] IBM developerWorks. Web services transaction (ws-transaction). *IBM developerWorks: Web services*, 2002. www-106.ibm.com/developerworks/webservices.

[Edm02] David Edmond. E-commerce technologies. Lecture 1-13, Queensland University of Technology, School of Information Systems, Brisbane, Australia, Semester 2 2002. Subject ITN262.

[Epi03] Epionet. *Web Services: A Business and Technical Guide*, 2003. www.epionet.com.

[Fer02] Dr. Eduardo B. Fernandez. Distributed object-oriented systems. Lecture 9, Florida Atlantic University, Department of Computer Science and Engineering, Florida, USA, Fall 2002. Subject COP6632.

[Fli03] Donald Flinn. What security concerns does ws-security address? *SearchWebServices.com: Ask the experts: Web Services Security*, December 2003. searchwebservices.techtarget.com/ateExperts.

[Gla01] Graham Glass. *Web Services: Building Blocks for Distributed Systems*. Prentice Hall PTR, 2001.

[Hel] Helios Software Solutions. *TextPad 4.7.2*. Downloadable from www.textpad.com.

[IT03] Andrew Ireland and Hamish Taylor. Distributed systems programming. Lecture 1 - 15, Heriot-Watt University, School of Mathematical and Computer Sciences, Edinburgh, Scotland, Term 2 2003. Subject F29NM1.

[Jen01] Dieter E. Jenz. Web services - a reality check. *WebServices.Org*, 2001. www.webservices.org.

[Jup04] Jupitermedia Corp. *Wĕbopēdia*, 2004. www.webopedia.com.

[Kru04] Philippe Kruchten. *The Rational Unified Process an Introduction*. Addison-Wesley, third edition, 2004.

[LE03] Olav Lysne and Frank Eliassen. Introduksjon til distribuerte system (ds). Lecture 1 - 11, University of Oslo, Department of Informatics, Oslo, Norway, autumn 2003. Subject INF5040, in Norwegian.

[Lou01] Steve Loughran. Modern development processes crises. Guest lecture, Oregon State University, Bristol, England, march 2001. www.iseran.com, www.oregonstate.edu.

[Mac03] Sean MacRoibeaird. Universal description, discovery & integration (uddi), an executive summary. *XML at Sun: Developer Connection*, December 2003. wwws.sun.com/software/xml/developers/uddi.

[Mic] Microsoft Corporation. *Windows XP Professional*. Product information on www.microsoft.com/windowsxp/pro.

[Min01] Nelson Minar. Distributed systems topologies: Part 1. *O'Reilly Network*, 2001. www.oreillynet.com.

[Min02] Nelson Minar. Distributed systems topologies: Part 2. *O'Reilly Network*, 2002. www.oreillynet.com.

[Mum01] David Alan Mumford. Jini and distributed systems resource monitoring. Final year project, University of Portsmouth, The Distributed Systems Group, Portsmouth, UK, 2001. Project unit: PJE40.

[Nat03] Maitreya Natu. Soap. Graduation project presentation 1, Deptartment of Computer and Information Science, Unversity of Delaware, Newark, USA, December 2003. Published on www.cis.udel.edu/~natu.

[OAS01] OASIS RELAX NG TC. *RELAX NG Specification*, December 2001. ww.oasis-open.org.

[Roe03] Assoc. Prof. Paul Roe. Software development for the web. Lecture 11 & 13, Queensland University of Technology, School of Software Engineering and Data Communications, Brisbane, Australia, Semester 1 2003. Subject ITN471.

[Sma03] Nigel Smart. *Cryptography: An Introduction*. McGraw-Hill Education, 2003.

[Suna] Sun Microsystems, Inc. *Java 2 Platform, Enterprise Edition (J2EE)*. Downloadable from java.sun.com/j2ee.

[Sunb] Sun Microsystems, Inc. *Java 2 Platform, Standard Edition (J2SE)*. Downloadable from java.sun.com/j2se.

[Sunc] Sun Microsystems, Inc. *Java Technology and Web Services*. Downloadable from java.sun.com/webservices.

[Sun04a] Sun Microsystems, Inc. *Dynamic code downloading using JavaTM RMI (Using the java.rmi.server.codebase Property)*, 2004. java.sun.com/j2se/1.5.0/docs.

[Sun04b] Sun Microsystems, Inc. *Java Technology*, 2004. www.sun.com.

[Sun04c] Sun Microsystems, Inc. *rmic - The Java RMI Compiler*, 2004. java.sun.com/j2se/1.5.0/docs.

[Tec04] TechTarget. *WhatIs.com*, 2004. whatis.techtarget.com.

[UDD03] UDDI Spec Technical Committee. *UDDI Version 3.0.1*, October 2003. uddi.org/pubs/uddi_v3.htm.

[VN02] Steven J. Vaughan-Nichols. Web services: Beyond the hype. *Computer*, Vol. 35, No. 2:18–21, February 2002. wwww.computer.org/computer.

[W3C03a] W3C Working Group. *SOAP Version 1.2 Part 0: Primer*, June 2003. www.w3.org/TR/soap12-part0.

[W3C03b] W3C Working Group. *SOAP Version 1.2 Part 1: Messaging Framework*, June 2003. www.w3.org/TR/soap12-part1.

[W3C03c] W3C Working Group. *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*, November 2003. www.w3.org/TR/wsdl20.

[W3C04a] W3C Working Group. *Extensible Markup Language (XML) 1.0 (Third Edition)*, February 2004. www.w3.org/TR/REC-xml.

[W3C04b] W3C Working Group. *Web Services Architecture*, February 2004. www.w3.org/TR/ws-arch.

[W3C04c] W3C Working Group. *Web Services Glossary*, February 2004. www.w3.org/TR/ws-gloss.

[WH03] Ian S. Welch and John H. Hine. Distributed systems. Lecture 1 - 24, Victoria University of Wellington, School of Mathematical and Computing Sciences, Wellington, New Zealand, Term 1 2003. Subject Comp 413.

[Won03] Dr. On Wong. Distributed systems. Lecture 1 - 12, Queensland University of Technology, School of Software Engineering & Data Communications, Brisbane, Australa, Semester 1 2003. Subject ITN484.

[YS02] Yehuda and Tomer Shiran. Web services, part i - xi. *Doc JavaScript*, Column 96 - 106, November 2001 - 2002. www.webreference.com.

Appendix A

Running the applications

This appendix describes how to run the various applications. This is also described on the enclosed CD.

My web service applications demand their own running server, I used the server Apache Tomcat 5.0.19 from the Apache Jakarta Project [Apaa], while my distributed applications provide their own server.

A.1 Poker

A.1.1 Original

To run the application: Simply start the client from the directory of the classes:

```
java PokerClient
```

A.1.2 Distributed

To run the application: First start the *rmi registry* and the server, then start the client. When starting the server and the client the server′ codebase and the policy file location must be specified. The code base location is typically `http://myhost/~myusrname/poker/` for this application.

Server side:
Windows: `start rmiregistry`
Solaris: `rmiregistry &`

```
java -Djava.rmi.server.codebase=<codebase>
-Djava.security.policy=<policy location> poker.PokerServer
```

Client side:

```
java -Djava.rmi.server.codebase=<codebase>
-Djava.security.policy=<policy location> poker.PokerClient
```

To compile the application: In case you may want to recompile the application.

Compile all the files:
```
javac -d <class files directory> IPoker.java Poker.java
PokerServer.java PokerClient.java
```

Compile the `Poker` class once more using the Java RMI stub compiler:
```
rmic poker.Poker
```

A.1.3 Web service

To run the application: The server must be started and the web service deployed before the web service can be accessed.
In Apache Tomcat this is done by first installing Tomcat and Apache Axis and copying the class file directory to the
`<Tomcat directory>\webapps\axis\WEB-INF\classes`.

Then the web service is deployed using a deployment descriptor while the Tomcat server is running:
```
java org.apache.axis.client.AdminClient pokerDeploy.wsdd
```
Undeployment is done by using a undeployment descriptor:
```
java org.apache.axis.client.AdminClient pokerUndeploy.wsdd
```

To run the client:
```
java PokerClient -l<service url>
```

To compile the application: In case you may want to recompile the application.
```
javac -d <class files directory> *.java
```

A.1.4 Hands examples

Example of hands that can be used when running the application:
```
3c,5h,As,5d,Kh
5s,5c,3s,Ad,Qs
6c,2c,Qc,4c,7c
7s,Js,8s,9d,Jc
```

A.2 Bulletin board

A.2.1 Distributed

To run the application: First start the *rmi registry* and the server, then start the client. When starting the server and the client the server′ codebase and the policy file location must be specified. The code base location is typically `http://myhost/~myusrname/bulletin/` for this application.

Server side:
Windows: `start rmiregistry`
Solaris: `rmiregistry &`

```
java -Djava.rmi.server.codebase=<codebase>
-Djava.security.policy=<policy location>
bulletin.BulletinBoardServer
```

Client side:

```
java -Djava.rmi.server.codebase=<codebase>
-Djava.security.policy=<policy location>
bulletin.BulletinBoardClient <nick>
```

To compile the application: In case you may want to recompile the application.

Compile all the files:

```
javac -d <class files directory> IMessage.java
IBulletinBoard.java BulletinBoard.java
BulletinBoardServer.java BulletinBoardClient.java
```

Compile the `Poker` class once more using the Java RMI stub compiler:

```
rmic bulletin.BulletinBoard
```

A.2.2 Web service

To run the application: The server must be started and the web service deployed before the web service can be accessed.
In Apache Tomcat this is done by first installing Tomcat and Apache Axis and copying the class file directory to the
`<Tomcat directory>\webapps\axis\WEB-INF\classes`.

Then the web service is deployed using a deployment descriptor while the Tomcat server is running:

```
java org.apache.axis.client.AdminClient bulletinDeploy.wsdd
```

Undeployment is done by using a undeployment descriptor:

```
java org.apache.axis.client.AdminClient bulletinUndeploy.wsdd
```

To run the client:

```
java BulletinClient -l<service url> <nick>
```

To compile the application: In case you may want to recompile the application.

```
javac -d <class files directory> *.java
```

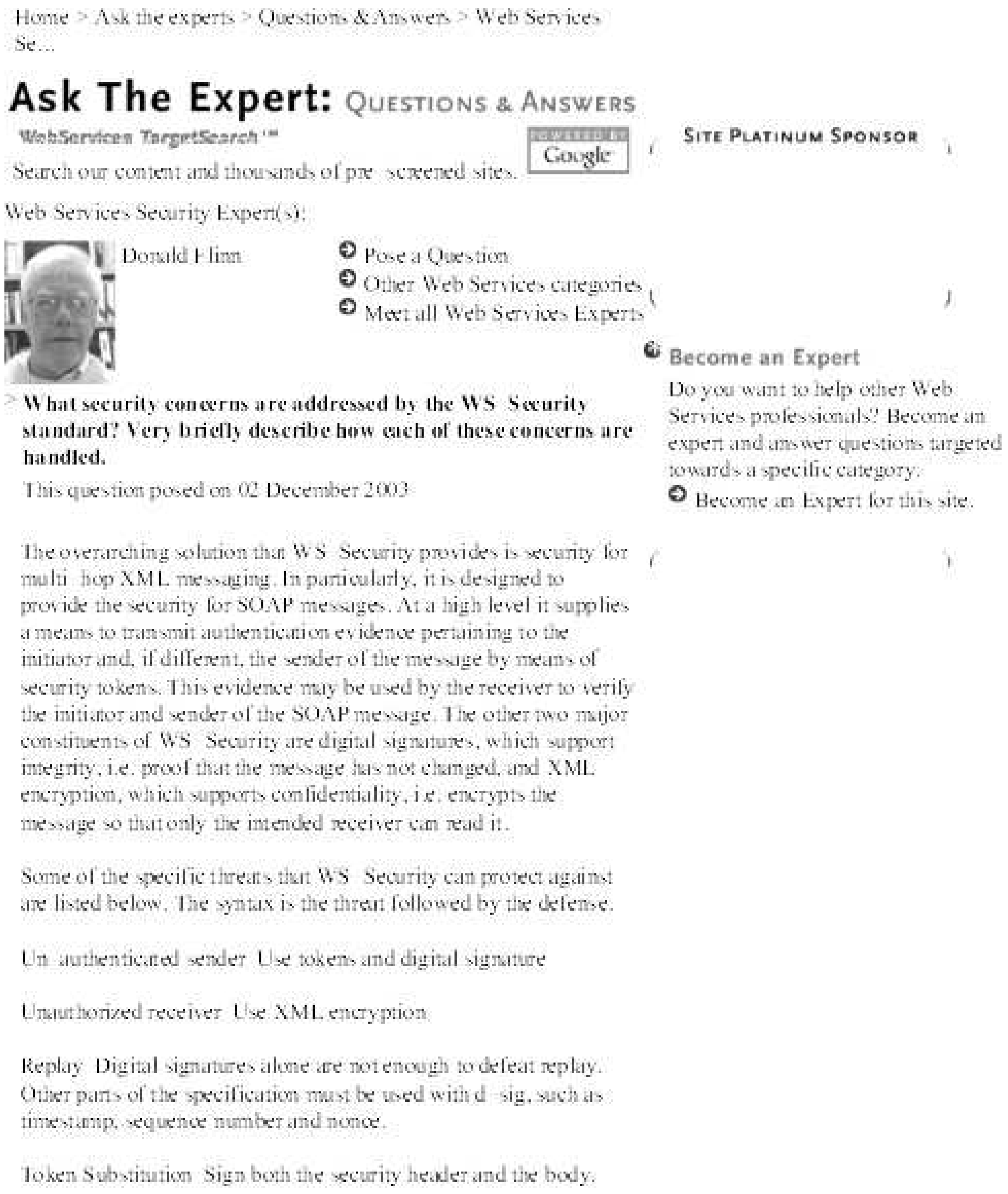
Appendix B

References

The reference sources I believe may be difficult to retrieve are listed in this appendix chapter. Other sources should be easy to access from the publisher. The reference sources in this appendix chapter are:

[Fli03]	Donald Flinn. What security concerns does WS-Security address?
[Nat03]	Maitreya Natu. SOAP
[VN02]	Steven J. Vaughan-Nichols. Web Services: Beyond the Hype

Donald Flinn. What security concerns does WS-Security address? [Fli03]



Home > Ask the experts > Questions & Answers > Web Services Se...

Ask The Expert: QUESTIONS & ANSWERS

WebServices TargetSearch™

Search our content and thousands of pre screened sites.

Powered by Google

SITE PLATINUM SPONSOR

Web Services Security Expert(s):

Donald Flinn

Pose a Question

Other Web Services categories

Meet all Web Services Experts

Become an Expert

Do you want to help other Web Services professionals? Become an expert and answer questions targeted towards a specific category.

Become an Expert for this site.

> **What security concerns are addressed by the WS Security standard? Very briefly describe how each of these concerns are handled.**

This question posed on 02 December 2003

The overarching solution that WS Security provides is security for multi hop XML messaging. In particularly, it is designed to provide the security for SOAP messages. At a high level it supplies a means to transmit authentication evidence pertaining to the initiator and, if different, the sender of the message by means of security tokens. This evidence may be used by the receiver to verify the initiator and sender of the SOAP message. The other two major constituents of WS Security are digital signatures, which support integrity, i.e. proof that the message has not changed, and XML encryption, which supports confidentiality, i.e. encrypts the message so that only the intended receiver can read it.

Some of the specific threats that WS Security can protect against are listed below. The syntax is the threat followed by the defense.

Un authenticated sender Use tokens and digital signature

Unauthorized receiver Use XML encryption

Replay Digital signatures alone are not enough to defeat replay. Other parts of the specification must be used with d sig, such as timestamp, sequence number and nonce.

Token Substitution Sign both the security header and the body.

Message modification Sign the message

Message substitution – Sign both the security header and message body

Man–in–the–middle Sign both the request and response

Multiple tokens using the same key Require that the token be included in WS Security header.

While WS Security provides the means to protect against these attacks, it is up to the users of WS–Security to apply the appropriate protections depending on the level of risk management required. For example, if a sender is requesting a casual stock quote they might not deem it necessary to use the above protection mechanisms. However, if they were buying a stock then they would want to protect against the above threats. The receiver of the request may have different risk requirements and thus require some of above mechanisms, which are not important to the sender. For example, for the request for a quote, they may require authentication and additionally may require different level of authentication for different value transactions.

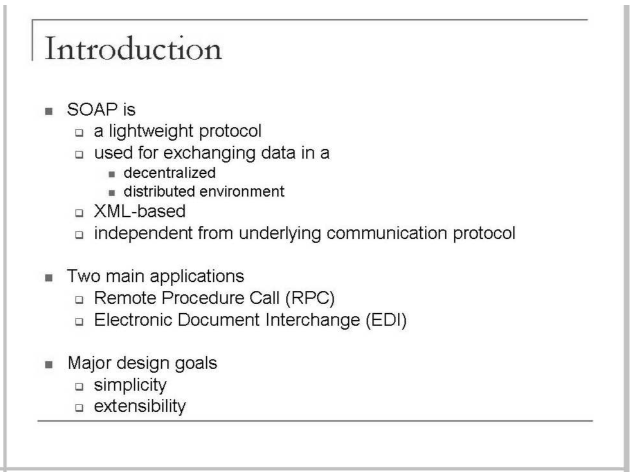
Go Back

Email a friend:

Note: Email addresses will only be used to send site content to your friend(s).

Maitreya Natu. SOAP [Nat03]

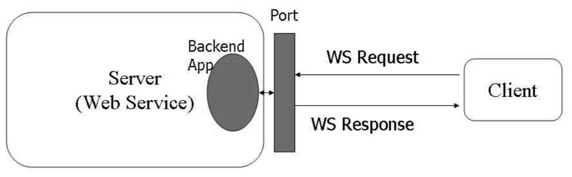
SOAP



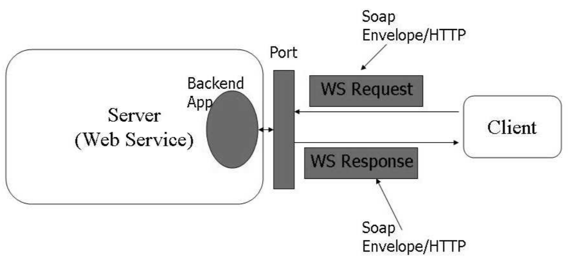
Introduction

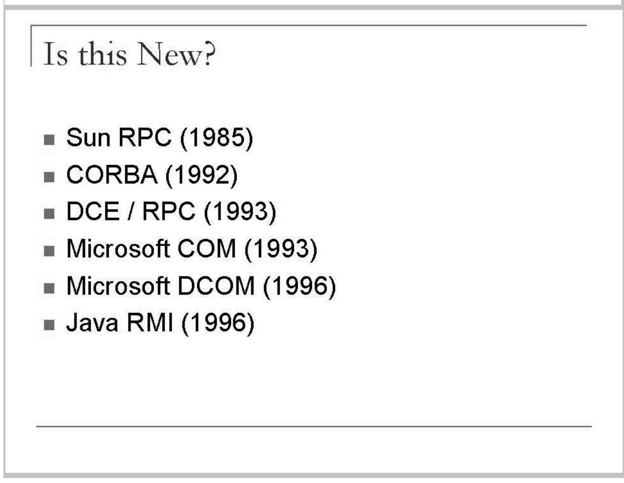
- SOAP is
  - a lightweight protocol
  - used for exchanging data in a
    - decentralized
    - distributed environment
  - XML-based
  - independent from underlying communication protocol
- Two main applications
  - Remote Procedure Call (RPC)
  - Electronic Document Interchange (EDI)
- Major design goals
  - simplicity
  - extensibility

Web Services Model



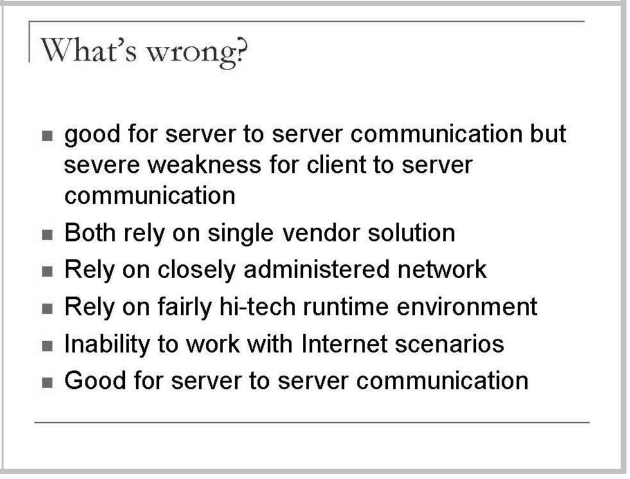
Web Services Model





Is this New?

- Sun RPC (1985)
- CORBA (1992)
- DCE / RPC (1993)
- Microsoft COM (1993)
- Microsoft DCOM (1996)
- Java RMI (1996)



What's wrong?

- good for server to server communication but severe weakness for client to server communication
- Both rely on single vendor solution
- Rely on closely administered network
- Rely on fairly hi-tech runtime environment
- Inability to work with Internet scenarios
- Good for server to server communication

Is this Different?

- Platform neutral
- Open Standards
  - Interoperable
- Based on ubiquitous software
  - XML Parsers
  - HTTP Server

HTTP+XML=SOAP

- HTTP as transport
  - SOAP Method: HTTP Request and Response that complies with SOAP encoding rules
  - SOAP EndPoint: HTTP based URI that identifies target for method invocation
  - other transport protocols can be used
- XML as payload
  - XML schema for serializing the data intended for packaging

HTTP (1/3)

- CORBA and DCOM for server to server and HTTP for Client to Server
- RPC Style Protocol: simple, widely deployed and likely to function in face of firewalls
- Handled by Web server software
- Request Response over TCP/IP
- Every HTTP server provides a much more efficient mechanism to get your code to process HTTP request
  - IS provides ASP and ISAPI
  - Apache allows C or PERL modules to run
  - Most application server modules allow you to write Servlets, COM components, EJB Session Beans, CORBA servants etc.

HTTP (2/3)

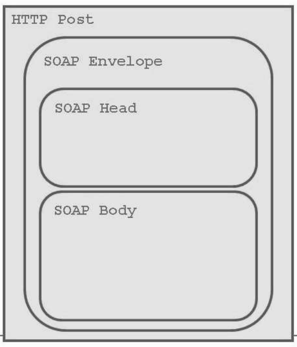
Example HTTP Request and Response

```
POST /foobar HTTP/1.1
Host: 209.110.197.12
Content-Type: text/plain
Content-Length: 12

Hello World
```

```
200 OK
Content-Type: text/plain
Content-Length: 12

dlrow ,olleH
```

```
400 Bad Request
Content-Length: 0
```

HTTP (3/3)



XML (1/5)

- XML is not a markup language. It's a set of rules for creating a new markup language such as HTML etc.
- XML: a simplified version of SGML to provide interoperability and scalability
- XML specifies specific syntax and semantic rules and constraints for creating new markup language

XML (2/5): Components of XML

- Elements: Lexical construct of a document
  - Contains content, either character data or other elements
  - <STREET> 4296 Razor Hill Road </STREET>
- Attributes: characteristics of an element
  - name/value pairs
  - <APPLET width="100" height="200">
- Comments: free text description
  - <!.....do not delete…….>
- Processing Instructions
  - To pass information to processing application
  - <?application data?>
- Document Type Definition:
  - Rules that govern your markup language
  - Declaration of element, element attributes, hierarchy of elements

XML (3/5): Schema and Instance

- Schema
  - Define the meaning, usage and relationships of the
    - Data-types elements and their contents
    - attributes and their values
    - entities and their contents
- Instance
  - An XML document whose structure conforms to some schema.

XML (4/5): Schema and Instance

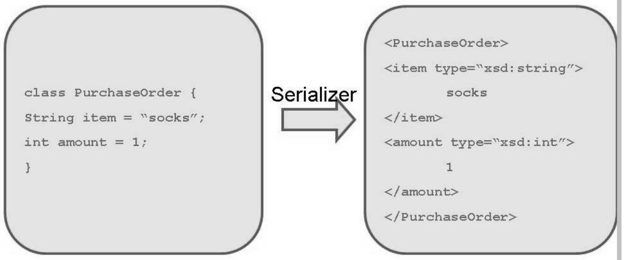
```
<schema
xmlns='http://www.w3.org/1999/XMLSchema'
targetNameSpace='urnschemas-develop-com:StringProcs'
>
 <element name='reverse_string'>
   <type>
     <element name='string1' type='string' />
     <element name='age' type='float' />
   </type>
 <element>
</schema>
```

Schema

```
<reverse_string
xmlns='urn:schemas-develop-som:StringProcs'
>
 <string1>Don Box</string1>
 <age>23.5</age>
</reverse_string>
```

Instance

XML (5/5): Serialization



```
class PurchaseOrder {
String item = "socks";
int amount = 1;
}
```

```
<PurchaseOrder>
<item type="xsd:string">
        socks
</item>
<amount type="xsd:int">
        1
</amount>
</PurchaseOrder>
```

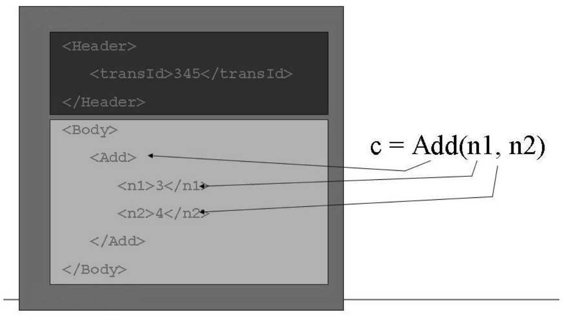
Packaging-SOAP : (1/6)

- Simple – The sending of xml over already well understood protocols is generally simpler than the alternatives
- Object – Comes its roots as a way to invoking COM Objects
- Access – Accessible design to allow it to be easy to make a web service
- Protocol – It defines how two nodes should communicate with each other.

Packaging-SOAP (2/6) : SOAP Elements

- Envelope (mandatory)
  - Top element of the XML document representing the message
- Header (optional)
  - Determines how a recipient of a SOAP message should process the message
  - Adds features to the SOAP message such as authentication, transaction management, payment, message routes, etc…
- Body (mandatory)
  - Exchanges information intended for the recipient of the message.
  - Typical use is for RPC calls and error reporting.

Packaging-SOAP (3/6) : Simple Example

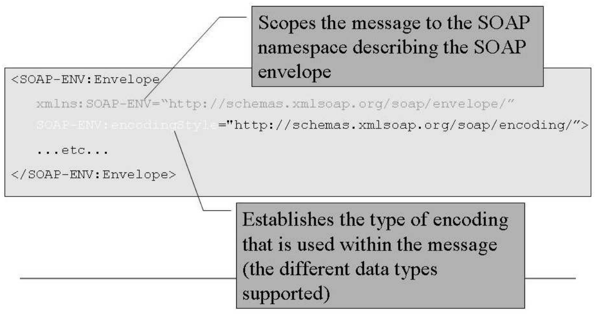


```
<Header>
    <transId>345</transId>
</Header>
<Body>
    <Add>
        <n1>3</n1>
        <n2>4</n2>
    </Add>
</Body>
```

Packaging-SOAP (4/6) : SOAP Request

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Header>
      <t:transId xmlns:t="http://a.com/trans">345</t:transId>
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
      <m:Add xmlns:m="http://a.com/Calculator">
         <n1>3</n1>
         <n2>4</n2>
      </m:Add>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Packaging-SOAP (4/6) : SOAP Request

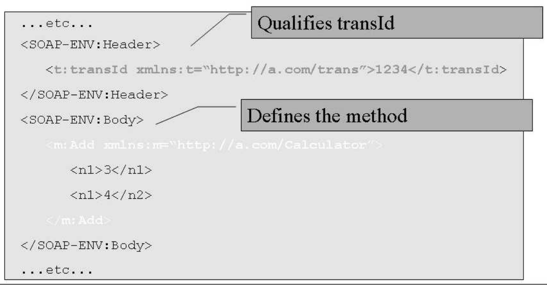


```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   ...etc...
</SOAP-ENV:Envelope>
```

Scopes the message to the SOAP namespace describing the SOAP envelope

Establishes the type of encoding that is used within the message (the different data types supported)

Packaging-SOAP (4/6) : SOAP Request



```
...etc...
<SOAP-ENV:Header>
   <t:transId xmlns:t="http://a.com/trans">1234</t:transId>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
   <m:Add xmlns:m="http://a.com/Calculator">
      <n1>3</n1>
      <n1>4</n2>
   </m:Add>
</SOAP-ENV:Body>
...etc...
```

Qualifies transId

Defines the method

Packaging-SOAP (5/6) : SOAP Response

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Header>
      <t:transId xmlns:t="http://a.com/trans">345</t:transId>
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
      <m:AddResponse xmlns:m="http://a.com/Calculator">
         <result>7</result>
      </m:AddResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Packaging-SOAP (5/6) : SOAP Response

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Header>
      <t:transId xmlns:t="http://a.com/trans">345</t:transId>
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
      <m:AddResponse xmlns:m="http://a.com/Calculator">
         <result>7</result>
      </m:AddResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Response typically uses method name with "Response" appended

Packaging-SOAP (6/6) : SOAP Fault

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Body>
      <SOAP-ENV:Fault>
         <faultcode>SOAP-ENV:Server</faultcode>
         <faultstring>Internal Application Error</faultstring>
         <detail xmlns:f="http://www.a.com/CalculatorFault">
            <f:errorCode>794634</f:errorCode>
            <f:errorMsg>Divide by zero</f:errorMsg>
         </detail>
      </SOAP-ENV:Fault>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP Encoding

- Method of serializing the data intended for packaging
  - mapping of basic application data types to XML format
  - allows applications to exchange data without knowing types a priori
  - An XML schema which is consistent with this type system can be constructed

SOAP Message Exchange Model

- A SOAP application on receiving a SOAP message must process that message by performing following actions:
  - Identify all parts of SOAP message intended for that application
  - Verify that all mandatory parts are supported by application and process them accordingly. If not, discard the message
  - Optional parts may be ignored
  - Remove all identified parts before forwarding the message
- Processing an application requires that SOAP processor understands message exchange pattern being used, role of recipient, employment of RPC (if any)

SOAP Processing Model (1/5)

- Describes the actions taken by a SOAP node on receiving a SOAP message
- The first step is overall check that the SOAP message is syntactically correct, i.e. conforms to processing instructions and DTD

SOAP Processing Model (2/5) : "role" attribute

- defines the (optional) role attribute that may be present in a header block
- It identifies the role played by the intended target of the header block.
- SOAP node is required to process a header block if it assumes the role identified by the value of the URI
- 3 standard roles have been identified
  - none
  - next
  - ultimateReceiver

SOAP Processing Model (3/5) : "mustunderstand" attribute

- To ensure that SOAP nodes do not ignore header blocks which are important to overall purpose of the application
- If "true", the node must process the block
- Processing of a message must not start until the node identifies all mandatory parts of the header blocks targeted to itself and understood them
- Node must be capable to process whatever is described in that block's specification or else discard it generating SOAP fault

SOAP Processing Model (4/5) : "relay" attribute

- Indicates whether a header block targeted at SOAP intermediary must be relayed if not processed
- So the incapable intermediaries should ignore and relay it (opposite to default rule), so that it could be available for those who are capable
- This can be done by mustUnderstand=false, relay=true

SOAP Processing Model (5/5)

```
<?xml version="1.0" ?>
  <env:Envelop xmlns:env="http://www.w3.org/2003/05/soap-envelop">
    <env:Header>
     <p:oneBlock xmlns:p="http://example.com"
            env:role="http://example.com/Log"
            env:mustUnderstand="true">
            ....
    </p:oneBlock>
    <q:anotherBlock xmlns:p=http://example.com
            env:role="http://www.w3.org/2003/05/soap-envelop/role/next"
            env:relay="true">
            ....
    </q:anotherBlock>

    <r:thirdBlock xmlns:p="http://example.com"
            ....
    </p:thirdBlock>
</env:Envelop>
```

SOAP Bindings (1/5)

- Specification of how SOAP messages may be passed from one SOAP node to another using an underlying protocol
- There can be different types of bindings
  - Pure XML
  - Compressed structure
  - Encrypted structure

SOAP Bindings (2/5) : SOAP Modules

- If a feature is not available through binding, it may be implemented with in the SOAP envelop using header blocks (identified by some URI) containing SOAP modules
  - If using UDP, SOAP module itself has to provide message correlation or directly the application should take care of it
  - If using HTTP as a protocol providing the service, the application could inherit this feature provided by binding and no further support is needed
- Any feature that is required by an application and may not be available can be carried as a SOAP module

SOAP Bindings (3/5) : Using SOAP with HTTP

- HTTP implicitly correlates request with response
- SOAP follows the HTTP request/response message model providing SOAP request parameters in HTTP request and response in HTTP response
- HTTP allows multiple intermediaries between client and server. However SOAP intermediaries are different from HTTP intermediaries.

SOAP Bindings (4/5) : HTTP Request

```
POST /Calculator.pl HTTP/1.0
Host: www.a.com
Accept: text/*
Content-type: text/xml
Content-length: nnnn
SOAPAction: "http://www.a.com/Calculator#Add"
{CR}{LF}
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Header>
      <t:transId xmlns:t="http://a.com/trans">345</t:transId>
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
      <m:Add xmlns:m="http://a.com/Calculator">
         <n1>3</n2>
         <n1>4</n2>
      </m:Add>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

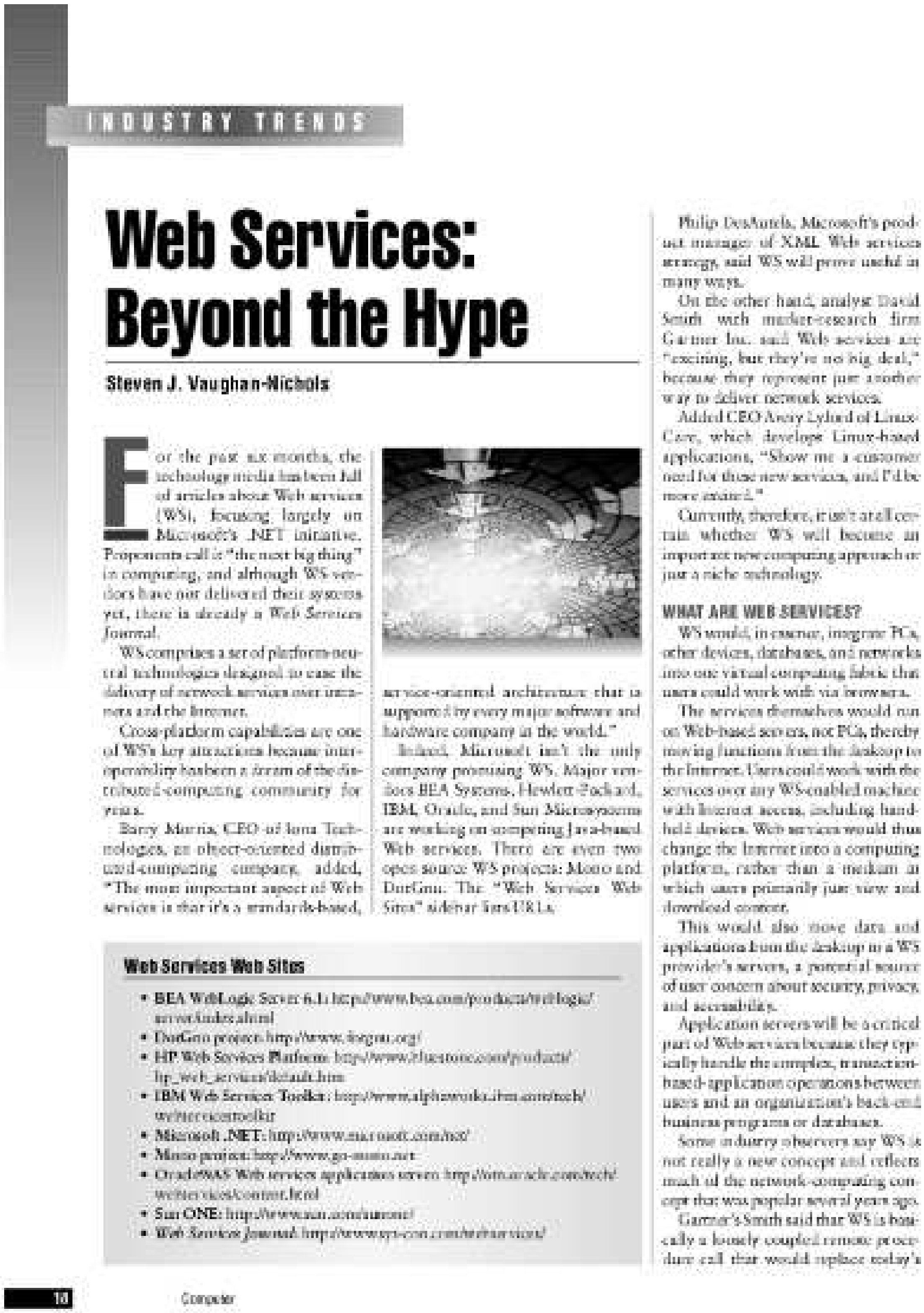
SOAP Bindings (5/5) : HTTP Response

```
HTTP/1.0 200 OK
Content-type: text/xml
Content-length: nnnn
{CR}{LF}
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Header>
      <t:transId xmlns:t="http://a.com/trans">345</t:transId>
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
      <m:AddResponse xmlns:m="http://a.com/Calculator">
         <result>7</result>
      </m:AddResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The Whole Picture

- UDDI - To discover where you can get web services and what businesses have to offer
- WSDL - To describe a web service and how to interact with it
- SOAP - To package your interaction with the Web Service
- HTTP Post -To carry the data envelope across the internet
- TCP/IP -To fragment and deliver the http post request to the end point

Steven J. Vaughan-Nichols. Web Services: Beyond the Hype [VN02]



INDUSTRY TRENDS

Web Services: Beyond the Hype

Steven J. Vaughan-Nichols

For the past six months, the technology media has been full of articles about Web services (WS), focusing largely on Microsoft's .NET initiative. Proponents call it "the next big thing" in computing, and although WS vendors have not delivered their systems yet, there is already a *Web Services Journal*.

WS comprises a set of platform-neutral technologies designed to ease the delivery of network services over intranets and the Internet.

Cross-platform capabilities are one of WS's key attractions because interoperability has been a dream of the distributed-computing community for years.

Barry Morris, CEO of Iona Technologies, an object-oriented distributed-computing company, added, "The most important aspect of Web services is that it's a standards-based, service-oriented architecture that is supported by every major software and hardware company in the world."

Indeed, Microsoft isn't the only company promising WS. Major vendors BEA Systems, Hewlett-Packard, IBM, Oracle, and Sun Microsystems are working on competing Java-based Web services. There are even two open source WS projects: Mono and DotGnu. The "Web Services Web Sites" sidebar lists URLs.

Philip DesAutels, Microsoft's product manager of XML Web services strategy, said WS will prove useful in many ways.

On the other hand, analyst David Smith with market-research firm Gartner Inc. said Web services are "exciting, but they're no big deal," because they represent just another way to deliver network services.

Added CEO Art Lyford of LinuxCare, which develops Linux-based applications, "Show me a customer need for these new services, and I'd be more excited."

Currently, therefore, it isn't at all certain whether WS will become an important new computing approach or just a niche technology.

WHAT ARE WEB SERVICES?

WS would, in essence, integrate PCs, other devices, databases, and networks into one virtual computing fabric that users could work with via browsers.

The services themselves would run on Web-based servers, not PCs, thereby moving functions from the desktop to the Internet. Users could work with the services over any WS-enabled machine with Internet access, including handheld devices. Web services would thus change the Internet into a computing platform, rather than a medium in which users primarily just view and download content.

This would also move data and applications from the desktop to a WS provider's servers, a potential source of user concern about security, privacy, and accessibility.

Application servers will be a critical part of Web services because they typically handle the complex, transaction-based application operations between users and an organization's back-end business programs or databases.

Some industry observers say WS is not really a new concept and reflects much of the network-computing concept that was popular several years ago.

Gartner's Smith said that WS is basically a loosely coupled remote procedure call that would replace today's

Web Services Web Sites

- **BEA WebLogic Server 6.1:** http://www.bea.com/products/weblogic/server/index.shtml
- **DotGnu project:** http://www.dotgnu.org/
- **HP Web Services Platform:** http://www.bluestone.com/products/hp_web_services/default.htm
- **IBM Web Services Toolkit:** http://www.alphaworks.ibm.com/tech/webservicestoolkit
- **Microsoft .NET:** http://www.microsoft.com/net/
- **Mono project:** http://www.go-mono.net
- **Oracle9iAS Web services applications server:** http://otn.oracle.com/tech/webservices/content.html
- **Sun ONE:** http://www.sun.com/sunone/
- ***Web Services Journal*:** http://www.sys-con.com/webservices/

tightly coupled RPCs, which require application- and protocol-specific application programming interface (API) connections. WS uses XML, rather than C or C++, to call procedures.

Still other experts say WS is just a type of middleware-based API, with XML providing the front end to Java 2 Platform, Enterprise Edition (J2EE) or .NET application servers. Like middleware, WS would link the application server and client programs.

STANDARDS: THE HEART OF WS

WS enables interoperability via a set of open standards, which distinguishes it from previous network services such as Corba's Internet Inter-ORB Protocol (IIOP).

XML is the most important Web-services standard and is the basis for the other WS standards. As a metalanguage, XML lets a set of users define its own markup tags. The tags provide information about the data in a document to users on most platforms. This permits cross-platform communications and also lets organizations integrate different data types within their systems.

Systems would use SOAP (simple object access protocol) to run WS applications. SOAP lets a program working in one operating system communicate with a program working in another by using HTTP and XML as information-exchange mechanisms. SOAP specifies how to encode an HTTP header and an XML file to achieve this interoperability. Thus, an operating system or browser will need only SOAP compatibility to work with any Web service.

The XML-based Web Services Description Language describes the online services a business offers. WSDL also helps users access a Web service by providing information such as the nature of its interface.

Finally, businesses use WSDL to list their Web services on the Internet in an XML-based registry based on the UDDI (universal description, discovery, and integration) protocol. UDDI lets companies find publicly available Web services on the Internet or corporate networks, as Figure 1 shows.

In essence, WS provides developers with a widely applicable API. Client-server systems use hard-coded interfaces and protocols between applications, noted Rick Caccia, director of product management for KnowNow, a WS and system-integration software company. This requires users to work with proprietary standards for each client-server software package.

Caccia said technologies based on Corba's object request brokers (ORBs) are a bit more flexible because they move the application coupling to a higher level. ORBs let programs treat blocks of code as objects without worrying about interior details.

However, explained Dan Gisolfi, solutions architect for IBM's jStart emerging-technologies team, "Vendors compete on ORB implementations and thus no [business] motivation exists ... to achieve [full] interoperability."

WS makes the process more abstract than ORBs by delivering an entire external service without users having to worry about moving between internal code blocks. The overall WS process would depend on several key elements, as Figure 2 shows.

WS FUNCTIONALITY

According to Annrai O'Toole, chair of WS-vendor Cape Clear Software, WS would permit the increased integration of online businesses. Businesses could thus use WS in many ways.

For example, software vendors could use WS to sell their applications over the Internet on a per-use or sub-

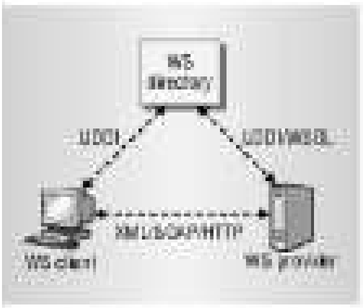


Figure 1. A Web services provider uses the universal description, discovery, and integration protocol to update the UDDI-based WS directory about the availability of its services. The provider sends the information encoded in WSDL. A client then sends a request for a service to the directory. The directory tells the client about the service's availability, and the client and provider interact via HTTP, SOAP, and XML.

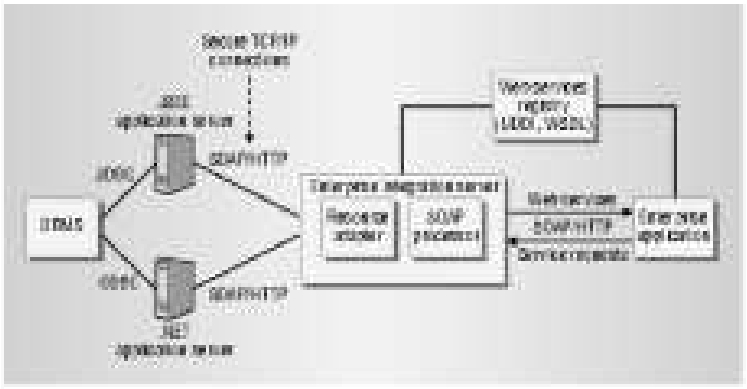
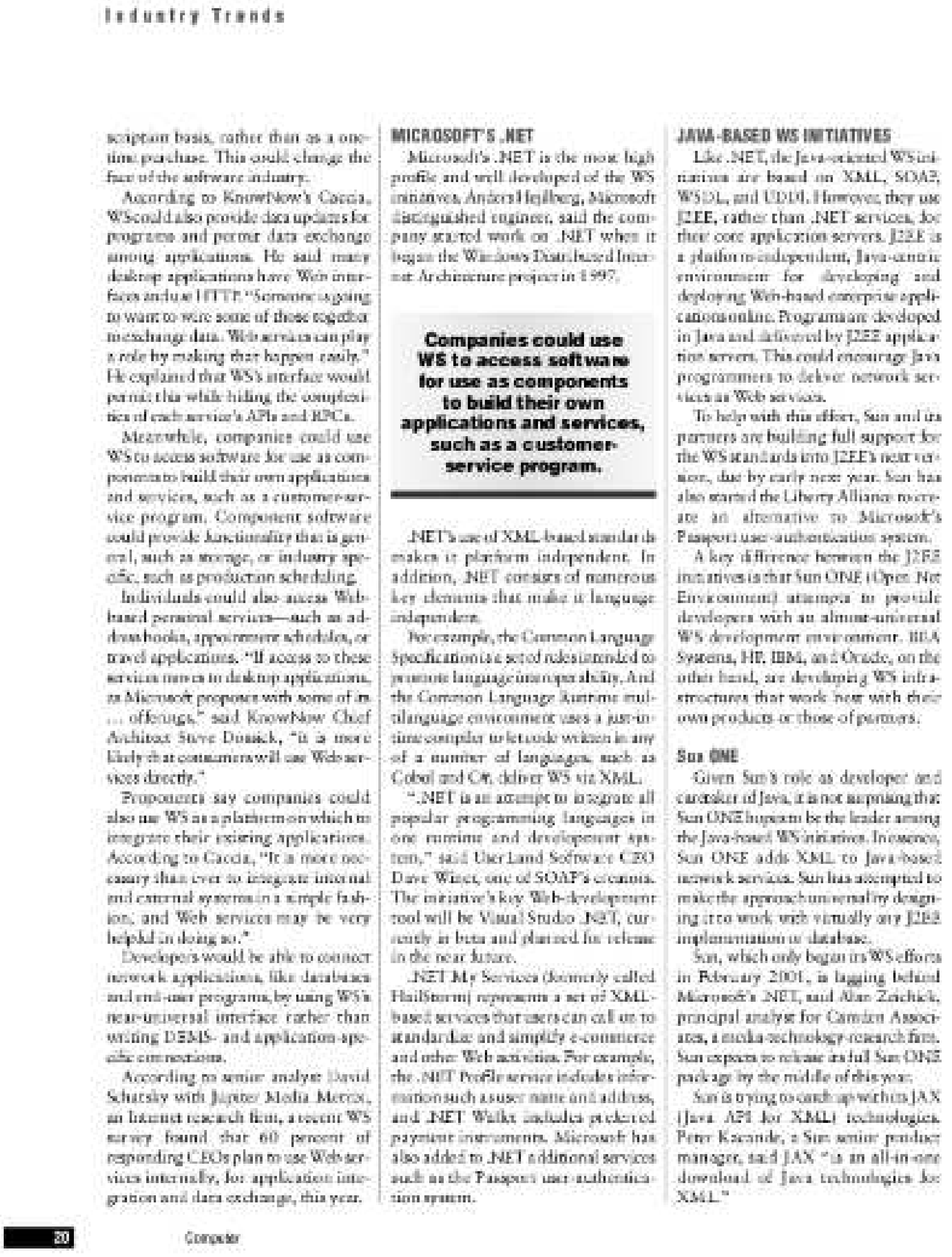


Figure 2. In a model Web-services (WS) process, an enterprise application requests a specific service, after determining its availability via the Web services registry, from a WS application server based on either Microsoft's .NET or J2EE. The resource adapter makes the connection between WS requests and the application server and also translates requests into specific network system calls. The processor translates SOAP into the appropriate requests for legacy servers and DBMSs that don't understand the protocol.



scription basis, rather than as a one-time purchase. This could change the face of the software industry.

According to KnowNow's Caccia, WS could also provide data updates for programs and permit data exchange among applications. He said many desktop applications have Web interfaces and use HTTP. "Someone is going to want to wire some of those together to exchange data. Web services can play a role by making that happen easily." He explained that WS's interface would permit this while hiding the complexities of each service's APIs and RPCs.

Meanwhile, companies could use WS to access software for use as components to build their own applications and services, such as a customer-service program. Component software could provide functionality that is general, such as storage, or industry specific, such as production scheduling.

Individuals could also access Web-based personal services—such as address books, appointment schedules, or travel applications. "If access to these services moves to desktop applications, as Microsoft proposes with some of its ... offerings," said KnowNow Chief Architect Steve Dossick, "it is more likely that consumers will use Web services directly."

Proponents say companies could also use WS as a platform on which to integrate their existing applications. According to Caccia, "It is more necessary than ever to integrate internal and external systems in a simple fashion, and Web services may be very helpful in doing so."

Developers would be able to connect network applications, like databases and end-user programs, by using WS's near-universal interface rather than writing DBMS- and application-specific connections.

According to senior analyst David Schatsky with Jupiter Media Metrix, an Internet research firm, a recent WS survey found that 60 percent of responding CEOs plan to use Web services internally, for application integration and data exchange, this year.

MICROSOFT'S .NET

Microsoft's .NET is the most high profile and well developed of the WS initiatives. Anders Hejlsberg, Microsoft distinguished engineer, said the company started work on .NET when it began the Windows Distributed Internet Architecture project in 1997.

Companies could use WS to access software for use as components to build their own applications and services, such as a customer-service program.

.NET's use of XML-based standards makes it platform independent. In addition, .NET consists of numerous key elements that make it language independent.

For example, the Common Language Specification is a set of rules intended to promote language interoperability. And the Common Language Runtime multilanguage environment uses a just-in-time compiler to let code written in any of a number of languages, such as Cobol and C#, deliver WS via XML.

".NET is an attempt to integrate all popular programming languages in one runtime and development system," said UserLand Software CEO Dave Winer, one of SOAP's creators. The initiative's key Web-development tool will be Visual Studio .NET, currently in beta and planned for release in the near future.

.NET My Services (formerly called HailStorm) represents a set of XML-based services that users can call on to standardize and simplify e-commerce and other Web activities. For example, the .NET Profile service includes information such as user name and address, and .NET Wallet includes preferred payment instruments. Microsoft has also added to .NET additional services such as the Passport user-authentication system.

JAVA-BASED WS INITIATIVES

Like .NET, the Java-oriented WS initiatives are based on XML, SOAP, WSDL, and UDDI. However, they use J2EE, rather than .NET services, for their core application servers. J2EE is a platform-independent, Java-centric environment for developing and deploying Web-based enterprise applications online. Programs are developed in Java and delivered by J2EE application servers. This could encourage Java programmers to deliver network services as Web services.

To help with this effort, Sun and its partners are building full support for the WS standards into J2EE's next version, due by early next year. Sun has also started the Liberty Alliance to create an alternative to Microsoft's Passport user-authentication system.

A key difference between the J2EE initiatives is that Sun ONE (Open Net Environment) attempts to provide developers with an almost-universal WS development environment. BEA Systems, HP, IBM, and Oracle, on the other hand, are developing WS infrastructures that work best with their own products or those of partners.

Sun ONE

Given Sun's role as developer and caretaker of Java, it is not surprising that Sun ONE hopes to be the leader among the Java-based WS initiatives. In essence, Sun ONE adds XML to Java-based network services. Sun has attempted to make the approach universal by designing it to work with virtually any J2EE implementation or database.

Sun, which only began its WS efforts in February 2001, is lagging behind Microsoft's .NET, said Alan Zeichick, principal analyst for Camden Associates, a media-technology-research firm. Sun expects to release its full Sun ONE package by the middle of this year.

Sun is trying to catch up with its JAX (Java API for XML) technologies. Peter Kacandes, a Sun senior product manager, said JAX "is an all-in-one download of Java technologies for XML."

JAX has several elements, including

- JAX-RPC, which lets developers build Web applications and services that incorporate XML-based RPCs via SOAP; and
- JAXP (Java API for XML processing), which would provide a standard way to integrate any XML-compliant parser with a Java-based application, thereby letting systems read, manipulate, and generate XML documents via Java APIs.

Sun ONE will work with the company's iPlanet application server and Forte development environment. Sun ONE will also provide Java-based software-development tools.

Other Java-based WS initiatives

Other companies' Java-based WS efforts, scheduled for release this year, are very similar. The key difference is that each uses its sponsoring company's own J2EE implementation and works best with its own DBMSs.

BEA. Already a leading application-server vendor, BEA is first out of the J2EE-enabled WS gate with its BEA WebLogic Server 6.1.

HP. The HP Web Services Platform initiative began with the open source e-Speak project, which originally was designed to deliver Web services using HP-created technologies. However, the rise of XML for delivering WS made e-Speak less important. Today, HP is focusing on delivering WS using XML-based specifications and the recently acquired Bluestone J2EE engine as the core application server.

IBM. IBM is adding WS to its WebSphere application-server suite via the Web Services Toolkit, a software development kit that includes a run-time environment, architectural blueprint, tools, components, a demo, and examples to help in designing and executing WS applications.

Oracle. Oracle says the company has its own WS offerings, though not under a banner initiative. Oracle says that building WS capabilities is more of an evolutionary process in software. Oracle can use its market-leading database technology in conjunction with its WS initiatives.

To help provide WS, the company is developing its Oracle9iAS Web Services application server. The company is also working on JDeveloper, a development kit that can be seen as a Java-based counterpart to Microsoft's Visual Studio.NET.

WS could help users save money by making it easier for them to develop and integrate their network applications.

CONCERNS

Industry observers, such as Steve Vinoski, Iona's chief architect and vice president of platform technologies, are worried about WS performance. One main reason is that XML, unlike binary-based IIOP, is text-based and thus entails more data for systems to process. XML therefore runs more slowly over HTTP. Adding a security protocol like Secure Socket Layer (SSL) would slow performance even more. This could make WS impractical for activities over low-bandwidth connections such as dial-up modems.

Because Web services have no built-in security model, they must rely on SSL, virtual private networks, or other external measures. In general, said Scott Dietzen, BEA Systems' chief technology officer, public-key infrastructure and SSL will provide sufficient security.

Nonetheless, vendors such as Netegrity, Oblix, and OpenNetwork Technologies are working on products to manage WS security via, for example, authentication and encryption.

In addition, the Organization for the Advancement of Structured Information Standards is developing the Security Assertion Markup Language, a vendor-neutral format for WS-transaction authentication. However, OASIS doesn't plan to approve SAML as a draft standard until the second half of this year.

WS: BUST OR BOOM?

With the exception of current J2EE application servers, no vendor is shipping WS tools. At most, the tools are in late beta.

While there are some publicly available WS applications (for a current list, see http://www.xmethods.net/), most are relatively trivial, such as zip code and MP3 finders.

That may change quickly, though. Gartner predicts that 75 percent of companies with more than $100 million in annual revenue will use WS by the middle of this year and that the technology will reach mainstream users by 2004.

Jupiter's Schatsky said WS could succeed because it will help users save money by making it easier for them to develop and integrate their network applications. In the process, .NET and J2EE vendors will hope to make money from sales of and support for their application servers, as well as application-integration services.

As for the marketplace battle, KnowNow's Dornick said, "Just as [Java] and [Microsoft] camps coexist today, they will continue to do so. While Microsoft has articulated an excellent packaged vision for Web-services creation and hosting that includes development tools and server support, it is likely that the J2EE camps will do so as well."

Nonetheless, KnowNow's Caccia was cautious about WS's future. "Right now we are in the easy, simple, pie-in-the-sky phase," he said. "The next year or two will bring out many hard obstacles." ■

Steven J. Vaughan-Nichols is a freelance technology writer based in Arden, North Carolina. Contact him at sjvn@vna1.com.

Editor: Lee Garber, Computer, 10662 Los Vaqueros Circle, PO Box 3014, Los Alamitos, CA 90720-1314; lgarber@computer.org