Table of Contents for Chapter 8

Part 2. Architect's Toolbox - 'bird's view of the terrain'

<<< … >>>

Chapter 8. Standards and Protocols

Out of all Standards and Protocols in existence, we selected some that likely will be of major concern to you as an Enterprise Architect, just to get you started.
This is not to say that you should not be concerned with any other Standards and Protocols. In a complex architecture, you may and will encounter some other standards, and may have to dig into them deep too.

Be prepared to watch this field constantly as new standards and their different (unfortunately!) implementations appear.

We have chosen to present some of the standards that especially relevant to modern distributed Enterprise Architectures – WWW, HTTP, Java, XML, Web Services to name a few.

OSI Model

The Open System Interconnect Reference Model (OSI) is the ISO 7498 standard and is widely accepted as a basis for the understanding of network protocol stacks.

OSI Model defines seven layers – from the physical network protocols at the lowest layer to the application protocols at the uppermost layer. Each layer provides functions for the implementation of the layer above, and relies on functions from the layer below.
From the logical point of view, each layer communicates directly with its peer layers on other nodes.

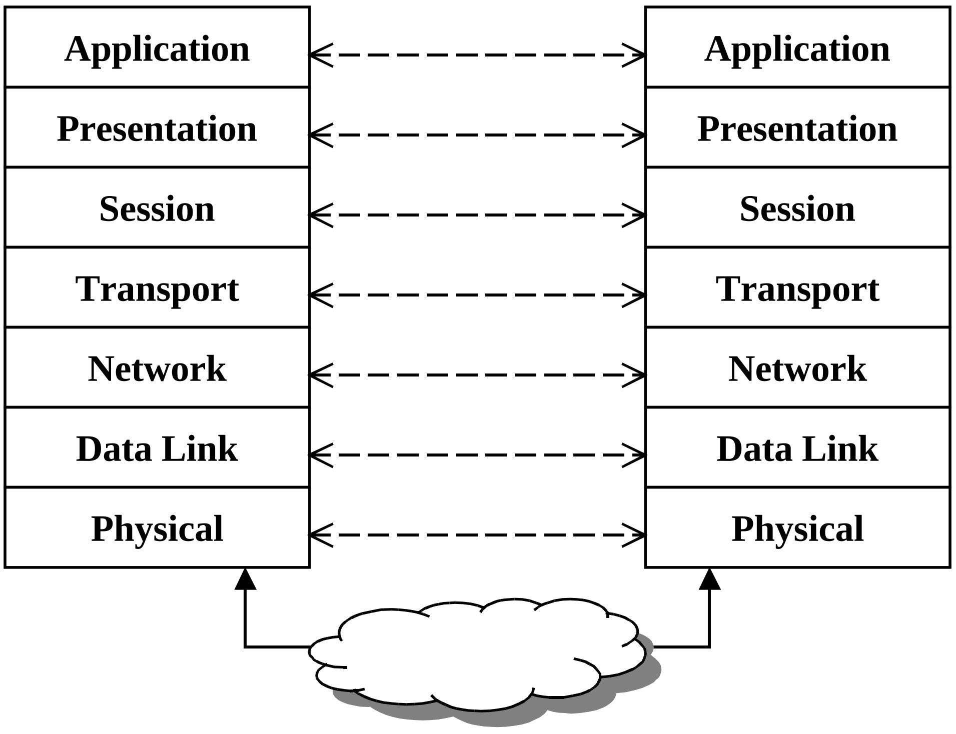


Figure 8.1. The OSI Reference Model

The seven layers are:

- ***Application*** – Network applications such as terminal emulation and file transfer
- ***Presentation*** – Formatting of data and encryption
- ***Session*** – Establishment and maintenance of sessions
- ***Transport*** – Provision of reliable and unreliable end-to-end delivery
- ***Network*** – Packet delivery, including routing
- ***Data Link*** – Framing of units of information and error checking
- ***Physical*** – Transmission of bits on physical hardware

TCP/IP – '… and family'

The *Transmission Control Protocol/Internet Protocol* (TCP/IP) is the common name for the protocol suite that has been universally accepted as a *lingua franca* for the Internet.
TCP/IP is the underlying highway for the networks worldwide and makes the true internetworking possible and ubiquitous.

Strictly speaking, we should use the term *Internet Protocol Suite*, which is used in official Internet standards.
Transmission Control Protocol (TCP) and *Internet Protocol* (IP) are two most important protocols in the suite of many. Short and catchy term TCP/IP is used to refer to the entire *Internet Protocol Suite*.

Figure 8.2 shows The TCP/IP Protocol Stack in relation to the networking layers in OSI and DARPA reference models. Not all protocols of the TCP/IP protocol suite are shown here.

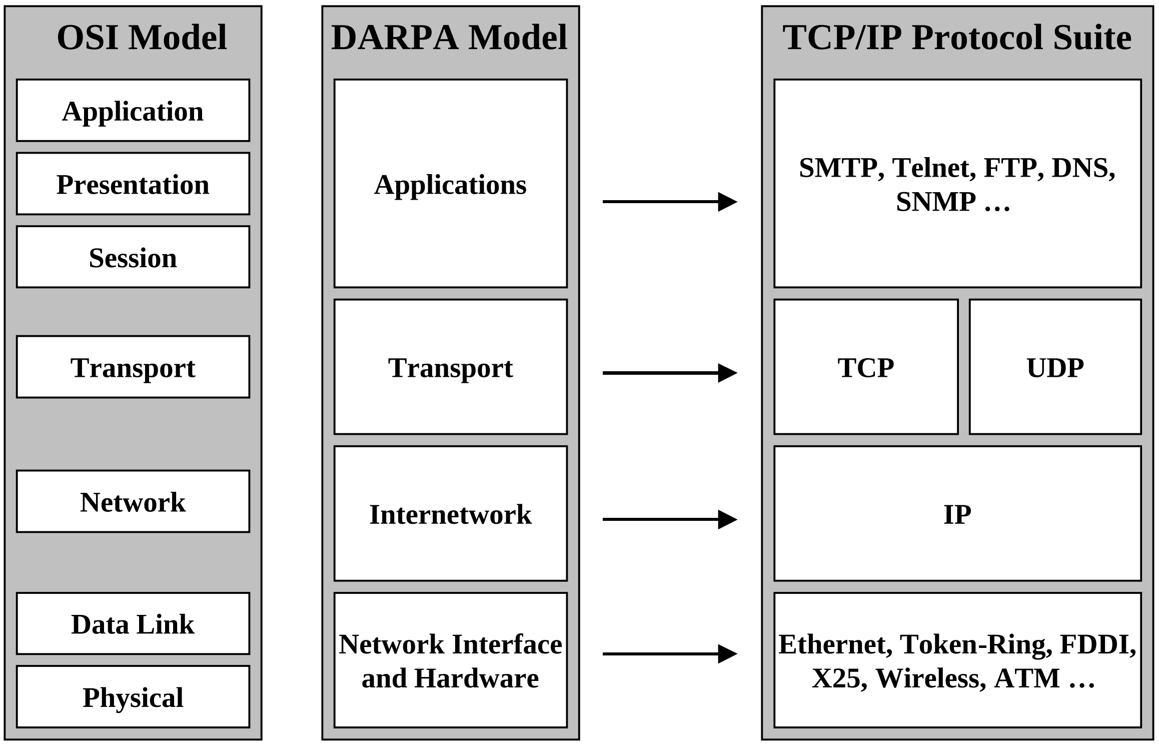


Figure 8.2. The TCP/IP Protocol Stack

Internet Protocol (IP)

Internetwork layer (DARPA) or Network layer (OSI), in addition to the *Internet Protocol* (IP), includes Internet Control Message Protocol (ICMP), Address Resolution Protocol (ARP), Reverse Address Resolution Protocol (RARP), Bootstrap Protocol (BOOTP) and Dynamic Host Configuration Protocol (DHCP).
These protocols perform datagram addressing, routing and delivery of datagrms, and resolve between the Internetwork layer addresses and Network Interface layer addresses.

Internet Protocol (IP) is a standard protocol STD 5 with the status of *required*.

IP hides the underlying physical network. IP is an unreliable, best-effort and connectionless protocol. Best-effort means that IP packets may be lost, or arrive out of order, or even be duplicated on arrival. IP assumes that these anomalies will be fixed in the higher-level protocols.

IP protocol uses IP addresses to uniquely identify any host on the Internet. Hosts exchange IP datagrams (or data packets) over the physical network they are connected to. Each IP datagram contains a *Source IP Address* and a *Destination IP Address*, and every datagram embarks on its own journey in the network from the source to the destination host, possibly visiting many places in the process.

IP addresses are represented by 32-bit unsigned binary value.

IP address is usually expressed in a dotted decimal format. By convention, 32-bit divided in 8-bit parts. IP address then represented as 4 dot-delimited decimal numbers, like 128.8.3.25.
The numeric form is used by IP software. The *Domain Name System* (DNS) provides mapping between IP addresses and more human-friendly symbolic name, like myhost.monin.com.

IP address consists of the pair of numbers – *Network Number* (or Network ID, or Network Address) and *Host Number* (on a given network). IP address uniquely identifies any host on the Internet. Regional Internet Registry (RIR, three of them) administers allocation of the *Network Number* portion of the IP address: American Registry for Internet Numbers (ARIN), European Registry (RIPE, from French) and Asia Pacific Network Information Centre (APNIC).

Network Number may take up 1, or 2, or 3 first bytes of the IP address – this determined by the Class of IP address.
In turn, the first 1, or 2, or 3 (and so on) bits of the IP address determine the Class of IP address (A, B, C, D or E) and how the rest of IP address should be interpreted.

Class D is reserved for multicast, and Class E – for the future use.
Figure 8.3 depicts layout of the Class A, B and C IP addresses.

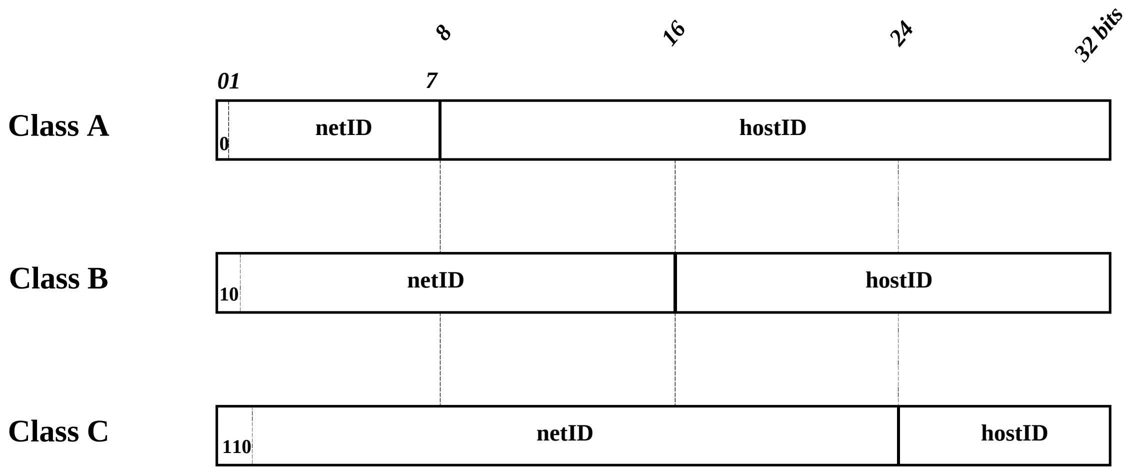


Figure 8.3. Classes of IP Addresses

Class A IP address uses 7 bits for the Network ID and 24 bits for the Host ID.
Class B address uses 14 bits for Network ID and 16 bits for Host ID. Class C uses 21 bits for Network ID and 8 bits for Host ID.
In general, Class A is used for the small number of very large networks, and Class C – for large number of small networks.

Some values of IP addresses are reserved and have a special meaning:

- All bits 0 in the Network ID or Host ID means *this* network or *this* host respectively
- All bits 1 interpreted as *all* networks or *all* hosts. For instance, directed broadcast to the whole network may be performed by providing valid Network ID and the Host ID all bits 1
- Address 127.0.0.0 is defined as a loopback network. Loopback interfaces do not access a physical network

With the growth of the Internet, the principle of assigned IP addresses became too inflexible for the network administration.
To avoid having to request additional IP network addresses, the concept of IP subnetting was introduced. The assignment of subnets is done locally, invisible to the world outside the network. This is done by subdividing the Host ID part of IP address into two parts – *subnetwork* (or *subnet*) and *host number*. As a result, Host ID is often called the *local address* or *local portion* of the IP address. The main network now consists of subnets that are managed by this network's administrator.

Un-assigned or un-allocated network numbers for the Class A, B and C networks are nearly exhausted. On one hand, strict control over allocated numbers was required. On other hand, we are running out of numbers, and need to relax the simplistic rule of IP addresses being globally unique.
Relief came from the introduction of Intranets with private IP addressing (when your host is not connected to the wider Interned, and no clashes of IP addresses occur) and the Classless Inter-Domain Routing (CIDR).

CIDR extends standard IP routing by ignoring the class bits of the network number and using all high order bits (IP prefix) of the IP address. CIDR uses network mask for combining multiple networks into a single entry – process called *supernetting*.

Transmission Control Protocol (TCP)

Transmission Control Protocol (TCP) is the most important and common protocol of the Transport layer of TCP/IP.

Another TCP/IP Transport layer protocol is the User Datagram Protocol (UDP) – thin unreliable protocol that does not provide a guaranteed delivery of packets, flow control, connections and error recovery. In essence, UDP is an application interface to IP.

TCP requires understanding of concept of *ports* (and, to lesser extent, of *sockets*).

Port is a TCP way of connecting the network to the process on the host that will be doing the actual job of handling the carried data.
Network talks to the *port* on the host with specified IP address; some process or program on the host is attached to the *port* and does actual talking on the host side using the known protocol.

Port is a 16-bit number. *Ports* may be of two types - *well known* or *ephemeral*.
Well-known ports range from 1 to 1023 and controlled by IANA. Well-known port numbers are defined in STD 2 – Assigned Internet Numbers. On most systems, well-known ports can be used only by system processes or by programs executed by privileged users. For instance, FTP uses ports 20 and 21; HTTP uses port 80 as default.
Ephemeral port numbers have values in the range 1024 to 65535 and can be used by ordinary user-developed programs.

Sockets are *de-facto* industry standard communication API introduced with BSD 4.2 Unix. Socket is a special type file handle that is used by process to request network services from the operating system. Using sockets, server processes are able to manage multiple conversations through a single port.

TCP is a standard protocol STD 7. TCP is a connection-oriented protocol that provides error recovery, flow control and reliable delivery of IP datagrams. TCP does not assume reliability of the underlying IP protocol; TCP guarantees reliability itself.
Most of the user application protocols, such as Telnet or FTP, use TCP.

To the application, TCP provides following services:

- ***Stream Data Transfer.*** Application views data as a contiguous stream. TCP bothers about splitting data into segments, transmitting them and assembling the data from segments on arrival
- ***Reliability.*** Each byte assigned the sequence number. Order of arriving segments is assured, missing segments detected by timeout and duplicate segments are eliminated. Time window is allocated to the transmission of certain number of bytes (window principle)
- ***Flow Control.*** TCP prevents overflow of its internal buffers (congestion control algorithm)
- ***Multiplexing.*** Achieved through the use of ports
- ***Logical Connections.*** TCP maintains certain status information for each data stream
- ***Full Duplex.*** TCP provides for concurrent data streams in both directions

TCP defines the segment format and the API to support this functionality.

WWW and Internet – 'cookies and tea'

"If you don't have a job that has a legitimate excuse for an Internet connection, change jobs immediately"
Adams, Scott. The Joy of Work: Dilbert's Guide to Finding Happiness at the Expense of Your Co-Workers. Harper Business, 1999

Internet captured imagination of the whole mankind and changed the way in which we entertain and infotain ourselves, and how we conduct our day-to-day business.
Internet has become a ubiquitous communication media and powerful equalizer that eliminated boundaries, distances and differences, and brought together multitude of vibrant virtual communities. Internet penetrated the very fabric of our society, business and mindset.

There are many ways to describe the impact of Internet on our lives – the light-hearted epigraph to this chapter above is as good as any, as it provides a pragmatic criteria or street-wise litmus test for your enterprise being in step with the technology.

Of course, we readily ignore the fact that Internet access may be abused for the personal use when on duty. But this is no different to the use of the telephone – each party need to be reasonable and responsible in their demands. There is communication media, and there are ways we use it.

In communication and networking terms, *Internet* means (and simply short for) *interconnected network* or *internetwork*, or universal worldwide communication services over heterogeneous physical networks.
Terms *Internet* and *WWW* (World Wide Web) may be used interchangeably.

Hypertext Transfer Protocol (HTTP) and Hypertext Markup Language (HTML)

Hypertext Transfer Protocol (HTTP) and Hypertext Markup Language are two separate Internet protocols.
HTTP is a communication and carrier protocol between HTTP client (Web browser) and HTTP server (Web host) on the Web.
HTML is a protocol for data presentation and rendering, data collection, and basic data processing.

In essence, HTML page is ASCII text program that will be parsed, interpreted and executed by some process understanding HTML. Therefore, HTML page ***is*** the executable program.
HTTP communication protocol carries HTML page as a payload from HTTP client to server, and back. HTTP provides context and parameters for the execution of the HTML page.
Both HTTP and HTML represent a ubiquitous *lingua franca* of the Internet, and usually used together.

HTTP

HTTP/1.1 has been standardised in RFC 2616 [WWW IETF]. World Wide Web Consortium [WWW W3C] developed HTML specifications.

HTTP is a simple *stateless* protocol that defines rules for the single request/response transaction with HTTP server. HTTP did a perfect job when Internet was in its infancy, when business was not so demanding, when researchers and scientists were totally happy just to communicate and to establish the web presence.
HTTP statelessness means that, unless web developers make a special programming effort, server would not know what happened before the current HTTP request, and what will happen after. This is what makes HTTP protocol simple, and, at the same time, causes so much grief in e-commerce applications, where business tries to implement complex business transactions, and to maintain high levels of security along the way.

Single HTTP transaction follows these steps:

- HTTP client (browser) opens the TCP connection to the Web host (HTTP server, or Web Server)
- Browser sends HTTP request to the host
- Server interprets the HTTP request, performs required actions and error processing, and composes the HTTP response
- Server sends HTTP response to the browser
- Browser closes connection

HTTP does not have a memory of the transaction state before or after this basic HTTP request/response transaction, and does not keep track of the connections. This means that every picture or other element on the HTML page may require a separate TCP connection for its download. However, HTTP/1.1 delivered a significant performance improvement by allowing persistent connections, thus reducing the amount of connections required. Persistent connections reduced latency in transmission and rendering the HTML page.

On Internet, there could be a long way from the browser to Web host. HTTP may require several intermediate connections on the way to the Web host, before it reaches the target Web Server pointed to by the HTTP URL.

Intermediate servers (proxies and gateways) may tunnel, or pass the content of HTTP request through untouched, or may cache the content to reduce the response time, or even perform more involved content tracking and manipulation. More about the adventures of HTTP request in transit later. For now, we assume that web browser connects directly to the target Web host.

HTTP provides a vehicle, carrier, or envelope for some useful HTML payload. HTTP also provides a feedback on how successful the transmission and transaction processing were.
HTTP defines a text-based format for the:

- HTTP request/response line with HTTP parameters like HTTP version, host URL, command
- HTTP request/response header or message meta-data, describing how to handle HTTP transaction, and what to expect in the content itself
- HTTP request/response body with the content of the message

While surfing the Net (short for Internet, or WWW), you may encounter some error conditions that will be presented to you as HTTP error or status codes.
Some of the more important HTTP status codes are:

- Informational (1xx)
- Successful (2xx)
    - 200 (OK)
    - 201 (Created)
    - 202 (Accepted)
    - 204 (No Content)
    - 206 (Partial Content)
- Redirection (3xx)
- Client Error (4xx)
    - 400 (Bad Request)
    - 401 (Unauthorized)
    - 404 (Not Found)
    - 408 (Request Timeout)
- Server Error (5xx)
    - 500 (Internal Server Error)
    - 503 (Service Unavailable)
    - 504 (Gateway Timeout)

Simplified HTTP request, and corresponding HTTP response (also simplified with some attributes omitted) may look like the ones on Figure 8.4:

HTTP request

```
GET / HTTP/1.1
Accept: image/gif, image/jpeg, application/msword
Accept-Language: en-us
User-Agent: Mozilla/4.0
Host: server12.research.safe-house.org:80
Connection: Keep-Alive
-- blank line --
```

HTTP response

```
HTTP/1.1 200 OK
Date: Mon, 30 Dec 2002 15:40:35 GMT
Server: Apache/1.3.12 (Unix)
Content-Length: 3680
Content-Type: text/html
-- blank line --
-- code of HTML page goes here --
```

© 2003 SAFE House

Figure 8.4. Simplified HTTP request and response

HTML

Hypertext Markup Language (HTML) Specification [WWW W3C] defines HTML as “the publishing language used by the World Wide Web”.

HTML is a text-based program that:

- formats and renders text and multimedia content
- provides links to other HTML pages on the Web and to the multimedia objects, and
- defines forms collecting the information from the Web surfer for the purpose of conducting business transaction, search, authentication etc.

“HTML was originally developed by Tim Berners-Lee while at CERN, and popularised by the Mosaic browser developed at NCSA” [WWW W3C, HTML Spec].

HTML achieves its goals by introducing set of formatting, information and control *tags*. It is possible to design HTML page using just a text editor, like in old days. However, nowadays web designer has plethora of WYSIWYG (What You See Is What You Get) web development tools to chose from, and can avoid delving into guts of HTML code. Still, understanding of HTML basics will help in developing efficient web pages.

Those familiar with basics of XML may note the resemblance. In terms of XML, HTML page *is* the fragment of XML code that is not guaranteed to be *well-formed*.

Reason for HTML being not a well-formed XML is that HTML is slack in allowing in some tags the absence of the closing tag. For instance, ‘*<p>*’ paragraph tag.
W3C tries to fix this problem by introducing and promoting XHTML (see section on XML) – the *well-formed* XML incarnation of the HTML. XHTML will aligh HTML evolution with other XML-related developments in the industry.

HTML page with complex formatting, and sloppy nesting and identation of tags may be very difficult to read. Try it yourself by clicking *View Source* on your browser’s menu bar.
However, code for the very simple HTML page may look like this:

```
<HTML>
    <HEAD>
        <TITLE>SAFE House</TITLE>
    </HEAD>
    <BODY>
        <P>Welcome to SAFE House Home Page!
    </BODY>
</HTML>
```

Figure 8.5. Simple HTML page

Browser and Web Server

Simply put, Web Browser and Web Server are respectively the client and the server end of the HTTP connection. Strictly speaking, they can talk other protocols like FTP or IIOP, but HTTP is by far is the prevalent way of communicating on the WWW.
If we ignore other protocols for the moment, we can use terms Web Server and HTTP Server interchangeably.

Web Browser and Web Server may run on different hardware and software. This does not matter, as long as they talk to each other over the Internet using HTTP protocol.
When we surf the WWW, we visit web sites that are implemented on vastly different platforms. Conversely, when we build web-based server application, we are able to reach clients over the Web on vastly different client PCs[1], or other Customer Premise Equipment (CPE), or otherwise web-enabled personal devices (like PDAs, mobile phones, palmtops etc).

[1] In reality, do not be surprised if the web client on PC that you are talking to runs some version of Microsoft Windows and Internet Explorer, or Netscape browser. Other desktop platforms are available, as well as evolving mobile and non-PC web clients.

Uniform Resource Locator (URL) – 'I'll find you …'

We commonly refer to WWW address as Uniform Resource Locator (URL).

RFC 2616 and RFC 2396 define syntax of URL and explain Uniform Resource Identifiers (URI) and Uniform Resource Names (URN). But use of the term URL shall suffice in most cases.

General form for the web address:

```
<protocol>":" "//" <host> [":"<port>] [<absolute_path>]
```

Specifically, HTTP URL according to RFC 2616 for HTTP/1.1:

```
http_URL = "http:" "//" host [":"port] [abs_path["?"query]]
```

'Query' in HTTP URL, if used, usually carries parameters of the HTTP request/response transaction between HTTP client and HTTP server in the form of name-value pair.

For instance, HTTP web address of the popular free mail service from Microsoft:

```
http://www.hotmail.com
```

The port number is optional. If port number is not specified, the default value for HTTP is 80.

Symbolic web address is resolved to the IP address of the server via Domain Name System (DNS). DNS is a standard protocol SDT 13 described in RFC 1034 and RFC 1035.

DNS provides a static implementation for the binding of URL to the IP address with little regard for security.
In order to take an advantage of Dynamic Host Configuration Protocol (DHCP) RFC 2131, a secure Dynamic Domain Name System (DDNS) is necessary. DHCP provides a framework for passing configuration information to hosts on a TCP/IP network.

Firewalls

A firewall is a part of networking infrastructure that enforces a security policy between a secure internal network and un-secure or untrusted public network, or Internet.

Firewall monitors and controls passing through it network traffic. Firewall is installed at a *choke* point where secure and untrusted networks meet, and is able to control both inbound and outbound traffic.

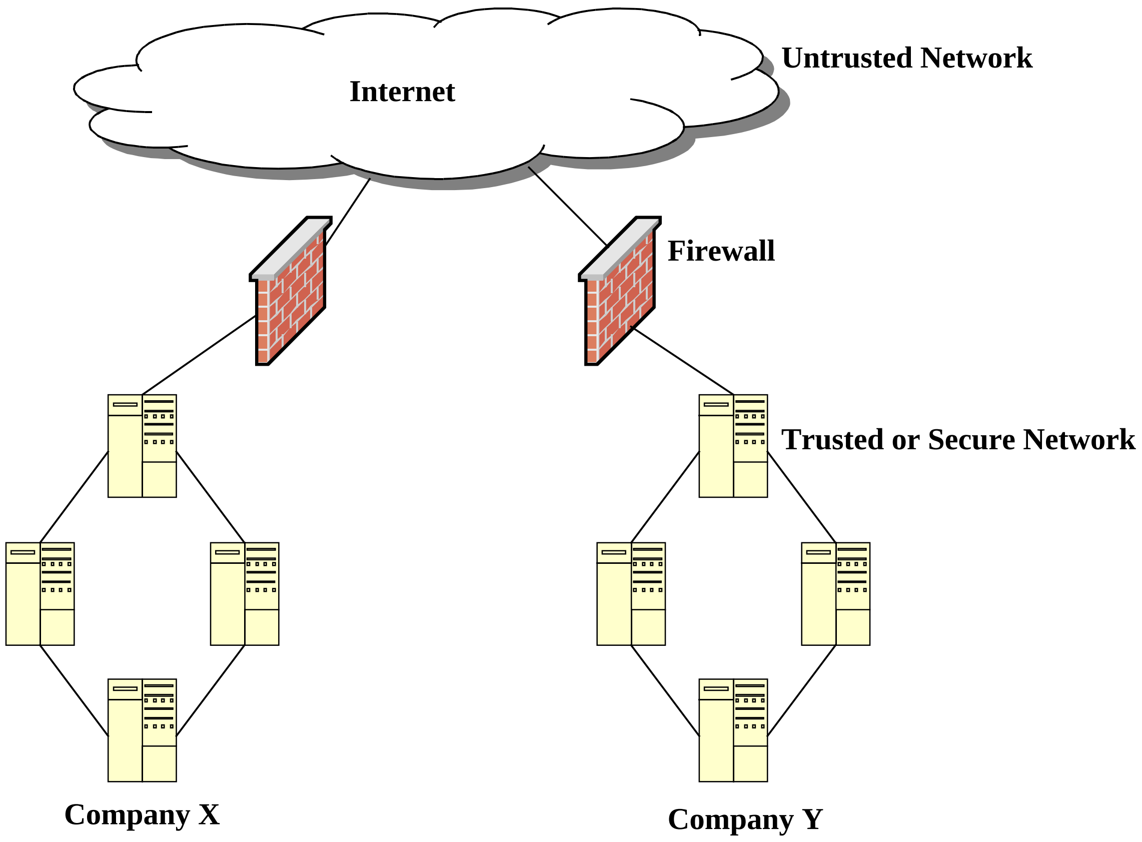


Figure 8.6. Firewall Concept

Firewall may consist of one or more functional components: packet-filtering router, application level gateway or proxy, and circuit level gateway.

Packet filtering is accomplished on the router by reading packet headers and applying some filtering rules. Router then makes the decision whether let packet to pass through or discard it.
Filtering rules may be built on the information from the packet header like source and destination IP address or port, message type and protocol.
Application level proxy controls and relays application-specific traffic, like HTTP, FTP, Telnet.
Circuit level gateways relay TCP connections and do not monitor or filter packets. Example of the circuit level gateway is SOCKS.

Packet filtering firewalls are very common, and are usually configured to deny any traffic that is not explicitly permitted.

The server that hosts the packet filtering router and the application level proxies is called the *bastion host*, because it is located squarely on the defence line between secure and untrusted networks. Bastion host facing two different networks is called the dual-homed gateway firewall, and will have at least two network interfaces and IP addresses.

Often firewall is configured with the bastion host and two packet filtering routers – outer router controls traffic from the untrusted network to the bastion host, and the inner router controls traffic between the bastion host and secure network. This firewall configuration creates *demilitarised zone* (DMZ) between external and internal networks.

Cookies

HTTP is a stateless protocol. Every HTTP transaction consists of HTTP request and response to and from HTTP Web Server, as if nothing else happened before or after the transaction.
This means that HTTP request/response transaction does not remember what happened in the previous HTTP transaction, unless we made a special effort to store the state and to make it available to the current HTTP request.

HTTP cookie is one of the methods to store and pass around the state of HTTP request/response transaction.

HTTP cookie is a single (for every domain) file of information that is stored on the client Web Browser, and exchanged between Web Browser and Web Server during HTTP transaction.

Maximum size of a cookie is 4K. Cookie contains text strings with name-value pairs of some parameters that represent the persistent state of the transaction and, hopefully, make sense to the application on the server.
Cookie is physically stored on the client Web Browser and plays the role of the (relatively small) scratchpad area for the prolonged conversation between the server application and Web Browser that spans several HTTP transactions.

Use of cookies is a common and legitimate technique of overcoming the statelessness of HTTP protocol. However, Enterprise Architect must be aware of the implications on the server applications when using cookies.

Cookies may be disabled on the browser, or physically deleted from the PC hard drive by the user. Or, enterprise security and privacy policies may prohibit the use of cookies.
If your application relies heavily on cookies to work, you may wish to rethink the business logic for the web access to your application. You may have to provide business logic and storage for persisting your transaction state on the server.
Contrary to the expected outcomes of no-fat diet, if you deprive yourself of cookies, you are likely to get fatter on the server.

Internet2

Before the commercial interests and the enthusiastic masses in general adopted Internet, research community enjoyed the exclusive use of the Internet. Not anymore – favourite toy was taken away. Furthermore, practical and architectural limitations of the Internet necessitated forward thinking on how we improve Internet so that it remains a dependable backbone for the information sharing in the society – be it for business or pleasure.

The university community, together with US government and industry partners, and with support of the Next Generation Internet (NGI) initiative, have formed the *Internet2* project [WWW, Internet2].
The NGI initiative is the US federal research program focusing on developing advanced networking technologies.

CORBA and MDA

The Common Object Request Broker Architecture (CORBA) [WWW OMG] is an open distributed object middleware, or message bus, standardized by the Object Management Group.

CORBA automates many integration, activation, location and networking tasks, primarily through location transparency and implementation transparency of objects, components, services, and facilities. CORBA distributed objects expose their services via object-oriented public interface, or contract, between the server object, and multitute of possible CORBA clients.

Public Interface of the CORBA object or service is defined using programming language-independent Interface Definition Language (IDL).

Figure 8.7 illustrates high-level CORBA Object Management Architecture.

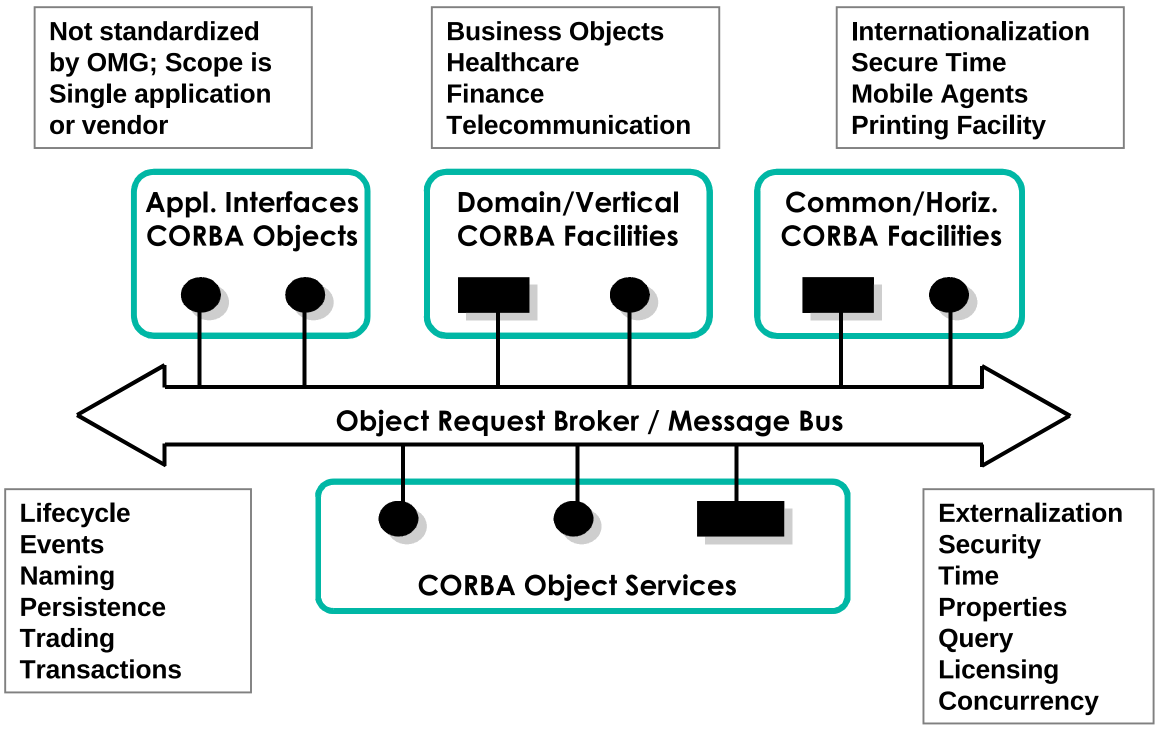
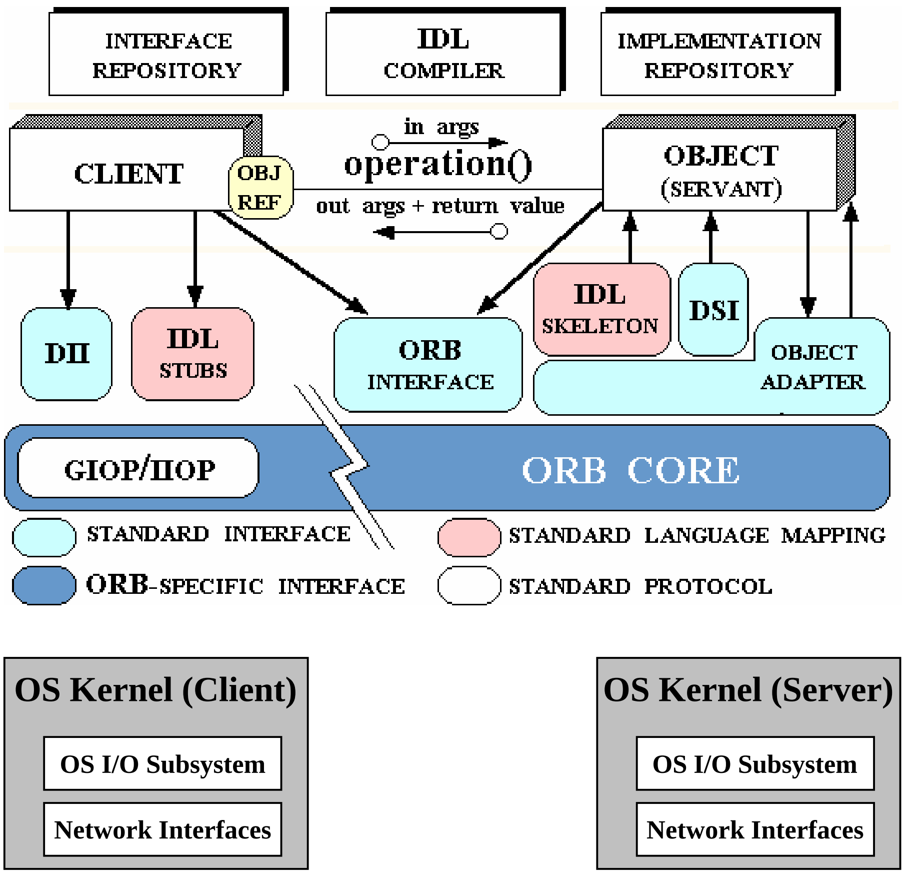


Figure 8.7. Object Management Architecture Overview

Figure 8.8 provides CORBA Reference Model outlining the essence of OMG middleware architecture.



Source: www.cs.wustl.edu/~schmidt/corba.html

© 2003 SAFE House

Figure 8.8. CORBA Reference Model

Multi-platform open ORB, and language-independent IDL Public Interfaces of the object insulate client and server from implementation specifics.
CORBA Object may run on different Operation System, re-written to different programming language, and CORBA client would not know the difference, provided that new server implementation still honours its contract in the form of IDL interface.

Servant is the specific implementation of the CORBA *Object*. Servants can be written in variety of programming languages, including C, C++, Java, Cobol, Ada, Smalltalk, Eifel, and more.

Client is the program entity that invokes an operationon the *Object* implementation. Client locates the required *Servant* object, and obtains its object reference. CORBA object reference is called Interoperable Object Reference (IOR) - symbolic pointer to the *Servant* that uniquely identifies to the ORB the location and instance of the server object implementation.
Before operation on the *Servant* can be executed, ORB makes sure that server object is activated, and *binds* client to the server.

IDL Stubs and *IDL Skeletons* produced by IDL Compiler specifically targeted for chosen programming languages for the client and servant respectively.
IDL Stubs for the specific language used by CORBA clients to marshal the operation and input arguments, and to unmarshal the output arguments and return codes when operation on server is completed.
IDL Skeletons for the specific programming language provided to the server object implementation for unmarshalling input arguments, and marshalling the output arguments and return codes back to the client.

Note that any combination of client and server supported, i.e. we can mix client and server implementations.

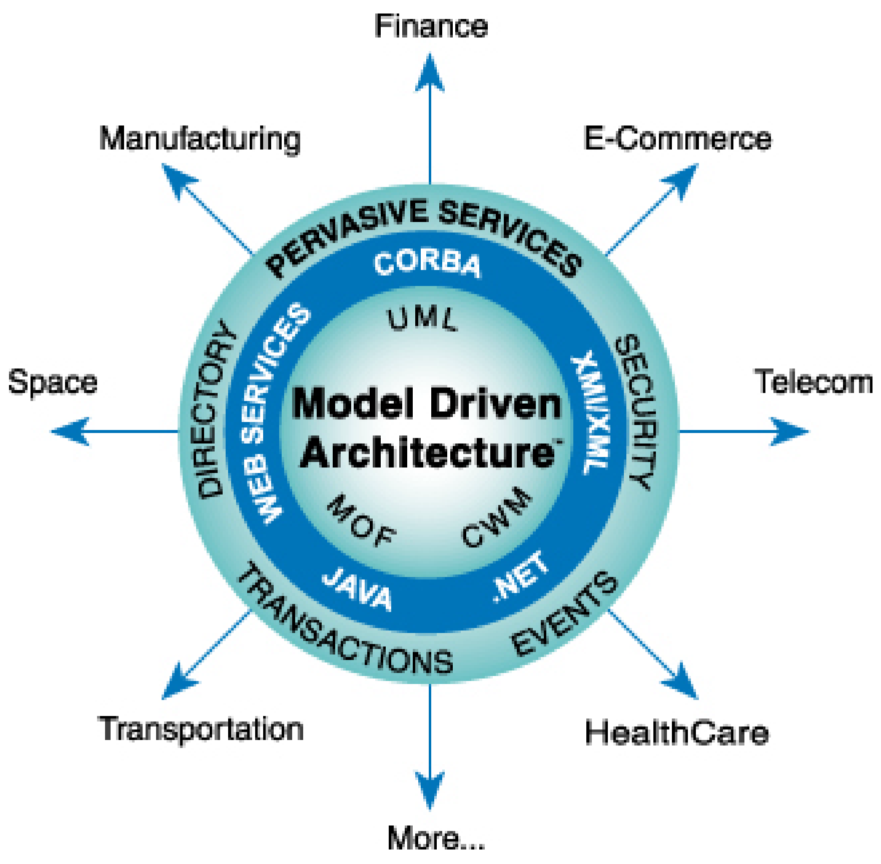
Dynamic Invocation Interface (DII) on the client side and the *Dynamic Skeleton Interface* (DSI) on the server side provide the *late binding* capability in CORBA, when interface may not be known at compile time.
Again, dynamic CORBA interfaces can be mixed in the same invocation with static RPC-style CORBA interfaces of IDL Stubs and Skeletons.

Object Adaptor activates the server object, and associates object implementations with the ORB. CORBA 3 standard introduced *Portable Object Adaptor* (POA) – highly configurable container for the server objects that ensures load balancing, scalability, and variety of servant activation policies.
In essence, POA is a comprehensive Object Transaction Processing Monitor in its own right.

OMG brings middleware standards one level further from CORBA by introducing the *Model Driven Architecture* (MDA).
MDA relies on rigorous capture and definition of the Enterprise Information Model in UML, complemented with model's metadata in XML – *Meta Object Facility* (MOF), *XML Metadata Interchange* (XMI), and *Common Warehouse Metamodel* (CWM, data warehouse standard).

MDA protects software investments by defining the Platform Independent Model (PIM), and mapping to the multiple *Platform Specific Models* (PSM) from the PIM. PSM can be potentially implemented in such middleware technologies like CORBA, DCOM, .NET, Java/EJB, Web Services with XML/SOAP, and other.
MDA tools aim to generate PSM from the PIM.

MDA is widely expected to facilitate integration of legacy applications and Commercial-Off-The-Shelf (COTS) products.



Source: OMG

Figure 8.9. OMG Model Driven Architecture

Extensible Markup Language (XML)

XML 101

The Extensible Markup Language (XML) is the language for defining a structured data.
Suite of XML standards is being developed and ratified by the World Wide Web Consortium (W3C, [WWW, W3C]).

XML is a blood relative of Structured Generalized Markup Language (SGML) and HyperText Markup Language (HTML).
SGML is an ISO 8897 standard for document definition in a high-end publishing. XML is a simplified subset of SGML (like HTML).
HTML is a simple markup language with pre-defined list of tags, and mixing information content and data presentation together.

XML clearly separates data and what we do with it in a sense of formatting and presentation - XML provides a clear separation between data and presentation (unlike HTML).

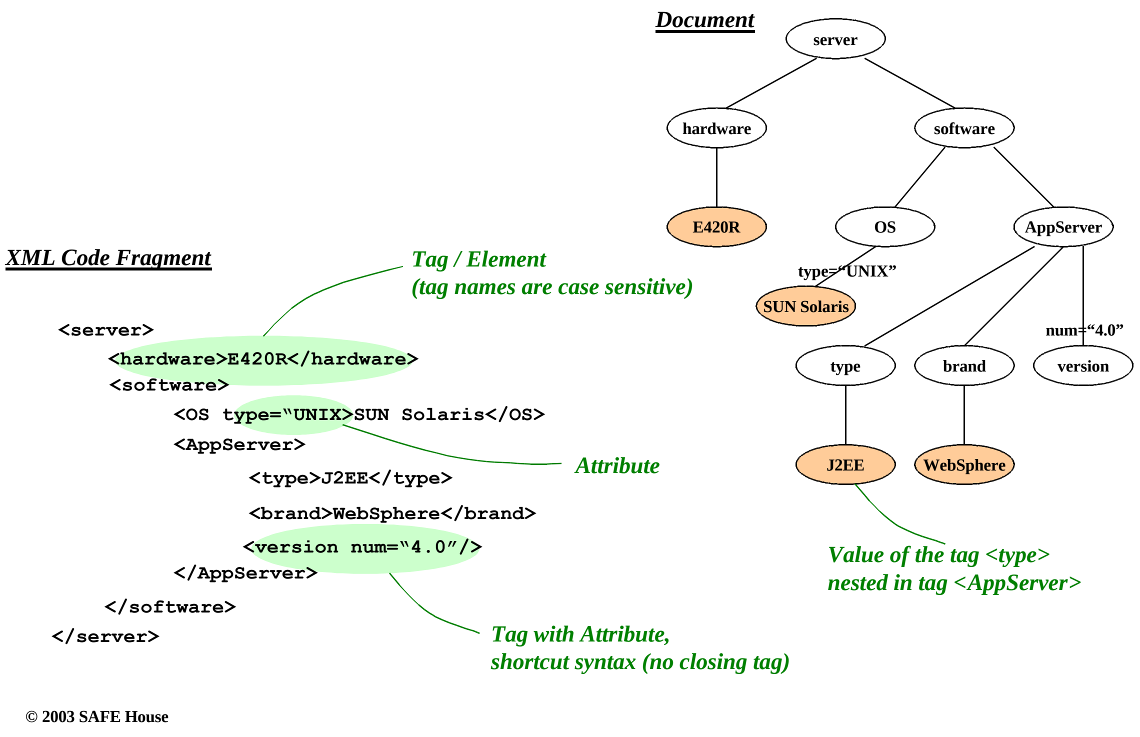
Another vital distinction of XML from HTML is that, strictly speaking, XML itself is not a language, but a *meta-language*. XML defines the syntax and framework for creating markup languages.

Before you use XML, you define your domain-specific vocabulary of tags and rules, and only then you endeavour to encode the instance of the document representing some portion of your information content.
XML is all about *syntax* – application will have to attach some *semantics* or meaning to the XML vocabulary.

Are you confused yet? Don't be. Fundamentals of XML are really simple, and are not going to change in essence with the evolution of still maturing standards. Last point itself makes worthwhile the efforts of understanding the XML basics, once and for all.

XML uses text tags (just like in HTML) to mark-up meaning of the content elements. XML document is a hierarchy of *Elements* that is represented by the nesting of XML tags.

Figure 8.10 provides an example of the fragment of the XML document.



```
<server>
    <hardware>E420R</hardware>
    <software>
         <OS type="UNIX>SUN Solaris</OS>
         <AppServer>
               <type>J2EE</type>
               <brand>WebSphere</brand>
               <version num="4.0"/>
         </AppServer>
    </software>
</server>
```

Figure 8.10. Sample XML Code

Figure demonstrates the various uses of tags for the document elements with nesting, with and without attributes and with shortcut syntax for tags.

Before we handle the XML document in the application, XML parser checks if document is syntaxically correct, or *well-formed*. I.e., all XML tags (*Elements* and *Attributes*) coded correctly, every tag provides opening and closing bracket for the *Element*, tags are not overlapping and tag nesting follows the strict hierarchy of *Elements* in the XML document.

If XML document is *well-formed*, we may proceed to checking if it is also *valid*, i.e. complies with the required document structure as defined in DTD or XML Schema as described in the next section.

By encoding the content of application objects in XML, we *serialise* or *marshal* document objects and pass them around as XML text strings. Or, rather, we *serialise* or *marshal* objects on the sending end of some message bus, and *de-serialise* and *de-marshal* objects on the receiving end.
We use XML to capture the structured information content, and for transforming, sharing and exchanging content between applications.

XML-based standards and vocabularies span many domains and categories, such as document schema definition and datatyping; document navigation; data transformation, re-purposing and formatting for presentation; middleware and technology-independent component integration; identity management; information modelling; access control and security; multimedia etc.
Sky is the limit for the purposeful utilisation of XML.

One of the applications of XML – XHTML, an XML vocabulary for HTML web pages, simply introduces more rigour into the structuring of HTML 4.0 code, and helps to minimise occurrences of confusion and errors in some browsers when they try to render seemingly correct web page. Missing HTML closing tag or overlapping HTML tags may work happily on some browsers, and fail on other. XHTML reduces possibility of HTML frivolity by enforcing the XML *well-formedness* on web page. XHTML is getting strong industry backing. However, it remains to be seen if mainstream web developers will adopt XHTML in favour of HTML 4.0. Apart from XHTML adoption and teething challenges, there is HTML legacy to deal with. XHTML will face an uphill battle for the universal acceptance. HTML, like COBOL of the Internet era, will stay with us for years to come.

Core standard of the XML suite of standards is the XML itself. XML 1.0 is a stable, widely adopted standard, ratified by the W3C in February 1998.

Original XML 1.0 Specification includes Document Type Definition (DTD) for the simple document structures. Use of DTD in the document is optional. However, if DTD is present, XML parser will use it on the *well-formed* (syntactically correct) document to check that it is *valid* (has got all required tags and attributes in place) as well.
DTD may be embedded into instance documents (Internal DTD) or placed on the Web and pointed to by provided URI for many documents to share (External DTD).

Figure 8.11 shows the sample DTD code that defines the document structure for the sample XML on Figure 8.10.

DTD Code Fragment

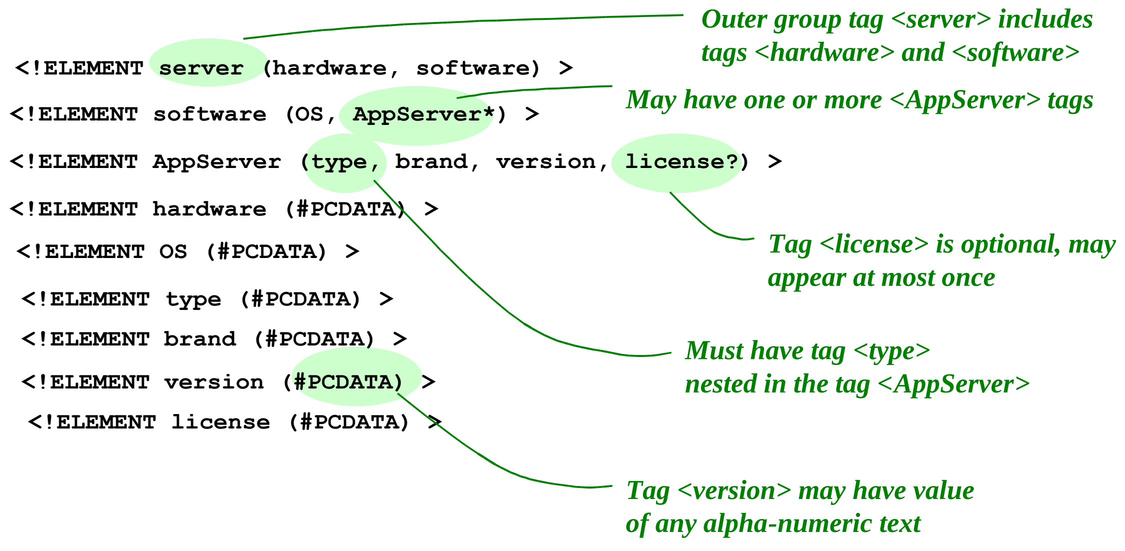

```
<!ELEMENT server (hardware, software) >
<!ELEMENT software (OS, AppServer*) >
<!ELEMENT AppServer (type, brand, version, license?) >
<!ELEMENT hardware (#PCDATA) >
<!ELEMENT OS (#PCDATA) >
<!ELEMENT type (#PCDATA) >
<!ELEMENT brand (#PCDATA) >
<!ELEMENT version (#PCDATA) >
<!ELEMENT license (#PCDATA) >
```
Outer group tag <server> includes tags <hardware> and <software>
May have one or more <AppServer> tags
Tag <license> is optional, may appear at most once
Must have tag <type> nested in the tag <AppServer>
Tag <version> may have value of any alpha-numeric text

Figure 8.11. Sample DTD

DTD defines the tags that can or must appear in the document, how the tags can be nested and required tag attributes.
DTD falls short of our expectations for flexibility of the XML document structure definition, on several counts:

- First thing you notice - DTD syntax is different from XML syntax
- No type support; or you can have value of any type, as long as it is string
- Limited set of rules for specific constraints; for instance, rules for enforcing cardinality, typing, data patterns
- Difficult to extend existing vocabularies

DTD is the part of core XML and will stay with us, but its deficiencies and peculiar syntax necessitated further improvements for the document definitions – through the development of XML Schema (see next section).

XML shifts the emphasis in middleware debacle from the hard choices between the more technology-oriented distributed platforms of DCOM and CORBA to the interoperation of loosely coupled and more technology-agnostic distributed objects.

XML suite of standards still rapidly evolves and matures. XML lives up to the promise of becoming the universal integration glue between components and applications. However, hype is still high. Range of available XML-based options and products becomes too confusing and at times intimidating, despite the simple rationale behind the XML frameworks.

Basic XML concepts in this section are less than 20% of XML specifications that will likely keep you afloat 80% of the time.

XML for Data Structuring, Transformations, Formatting and Semantics

Main ‘supplementary core’ XML standards are:

- ***XML Namespaces.*** As name implies, standard provides a mechanism for resolving XML tag and attribute name conflicts using namespace prefix with the tag names
- ***XML Schema.*** Specifies XML Schema definition language for document structure and datatypes. Is a superset of DTD, and replaces DTD. More datatypes than DTD (44+ versus 10), data patterns and object-oriented’ish type definitions and derivations. XML Schema defined in the same XML syntax as instance documents. Standard includes three parts – Part 0 (XML Schema Primer), Part 1 (Structures) and Part 2 (Datatypes)
- ***XSL and XSLT.*** Extensible Stylesheet Language (XSL) and supplementary XSL Transformations (XSLT) enable flexible formatting and rendering of XML data. XSL includes three parts – XSLT transformation language, Xpath (see below) that XSLT relies on, and Formatting Objects (XML vocabulary for display formatting, similar in concept to CSS and HTML). XSLT defines how to transform one document that is defined trough the DTD or schema (the “source tree”) into another XML document type (the “result tree”). Resulting document that is emitted by XSLT processor can have XML, HTML, or even non-XML format
- ***XPath.*** Specifies language for addressing nodes in an XML document tree. Syntax is not XML so that it is usable in attributes and URIs. Used by XSLT and XPointer
- ***XLink.*** Specifies XML elements for links between documents. Defines simple unidirectional hyperlinks similar to those in HTML, as well as more sophisticated multi-ended and typed links, with behavior of the remote resource targeted by the link
- ***XPointer.*** Based on XPath. Allows for examination of the document structure and choice based on content. Can address points and ranges as well as whole nodes. Locates information by string matching. Can define both absolute and relative locations in the XML document (relative to a particular structural element)

- ***DOM.*** Document Object Model (DOM) defines interface and binding to various programming languages and OMG IDL, that applications use to manipulate XML document tree in memory. Standard includes DOM Level 1, 2 and 3

Document Object Model (DOM) and Simple API for XML (SAX)

DOM SAX. No, that's not true. But we could not resist the temptation.

DOM is a *de-jure* W3C standard software API for parsing and manipulating the XML document tree in memory.
SAX is a *de-facto* XML standard software API for efficient parsing of XML documents.

DOM and SAX are very different in their features, but both are perfectly valid options for your XML development, depending on what you are trying to achieve.

Let us summarise the distinctive features of DOM and SAX so that you can make an informed choice of the API most suitable to your project (refer Table 8.1).

DOM	***SAX***
W3C Recommendation, *de-jure* standard	Industry *de-facto* broad acceptance
Object-Based Model – parser builds a DOM tree	Event-Based Model – parser sends events
Requires whole XML document tree structure in memory	Parses XML document as it comes, *interpreter* style. No state memory – event is forgotten after it fired
Requires more memory and grunt	Good if memory is limited
May be overkill for large XML documents and simple operations on them	Good if large document, and only parts of the document are required
Allows to edit the document	Good if you do not need to edit document
Good if you need to parse document multiple times	If you need to parse document again or go back – start again. SAX cannot navigate backwards in the document
Higher level API	Lower level API – gives you more control than DOM for memory utilisation, data structures and control flow in your application
Object-Based interface another level of indirection and hides parsing complexities	Event-Based interface is more natural for parser. Some DOM parsers are built on top of SAX parser
Requires less programmer's work	More work than DOM. Large and complex document, and intricate application business logic can get things out of hand quickly
May be less efficient	Almost always more efficient than DOM
Simple to learn and use as it closely matches the underlying XML document	Requires more involved document processing design and programming
Binding for many programming languages	Binding for many programming languages
Popular with scripting languages	Necessitates more powerful object-based programming languages – Java, Python, Perl, C++ etc
Preferred API in a browser	Popular in a server-side document processing

Table 8.1. DOM and SAX head to head

So, which XML document manipulation API is better relative to your specific requirements, DOM or SAX? Note, neither SAX or DOM is intrinsically better – they serve different needs.
Table 8.1 should give you pretty good idea for your evaluation, and well substantiated executive decision.

XML this, XML that

That's all, folks (quoting again our favourite rabbit), about the essence and fundamental concepts of XML. Well, not quite all, but almost.

This section sufficiently exhausted fundamental XML features that form the basis for your understanding of further developments in XML, related to more advanced definition of data structures, formatting and transformations of XML documents, and to the use of XML for interoperability and middleware connectivity.

Complete suite of XML standards and vocabularies, with all their features and rigour may seem overwhelming. Rest assured that not all XML standards are mentioned here – we've made a judgement call selecting most mature, stable and important ones.

Notation and syntax of standards and of the XML instance documents themselves are very terse and not very friendly to the human eye. You will be right to expect helpful visual tools, utilities and wizards to become available soon – they are appearing, improving, and more of these tools coming.

Despite XML being a (rather verbose and cryptic) text-based code, it was meant for machines, not for humans. XML programmers will not be left struggling with low-level XML complexities for long. After all, how many programmers can read a hexadecimal dump these days – but sometime back in the last millennium programmers had to write programs like that, and assembler seemed as a vast improvement. This reminder will cheer you up, hopefully.
And, while we distracted ourselves with analogies and ancient history of computing, there is another thought for you – VW beetle will seem very spacious after you tried to squeeze ten of your friends in the car first. Nothing will make you more appreciative than the experience that was worse…

XML is a long-anticipated enabling integration technology. XML ideas (under different names) were brewing for some time, and it was right time for XML to spill into IT mainstream as the raw computer power grew and the Enterprise Architecture demanded universal integration glue.

Armed with understanding of XML basics, you won't get lost in the discussions of DOM, Web Services, Common Warehouse Model (CWM), Common Identity Model (CIM), and numerous other applications of XML.

Web Services

Web Services 101

Web Services is the XML-based integration framework that provides a common interoperability middleware platform for connecting heterogeneous clients, servers, and services registries, possibly running on different hardware, operating systems, and implemented using different development or runtime environments.

Web Services elevates to the higher level of abstraction and commonality (or, heterogenity) that no middleware platform was able to enjoy before, and with much greater ease and simplicity. While assessing the Web Services model, compare it with other middleware infrastructures for heterogenity and simplicity, like CORBA, DCOM, DCE, and message queues.
Web Services framework provides glue between loosely coupled processes over the Internet.

Web Services framework has been defined in a suite of several related standards. Web Services specifications still evolve and expand, mainly towards better security, reliability, transactions, workflow management, and system management.
However, three core specifications that comprise the foundation of Web Services, have reached a degree of maturity in standards, their broad acceptance in the industry, and their available interoperable implementations.

Core Web Services standards are:

- ***SOAP*** – Simple Object Access Protocol. XML-based mechanism for sending the message (or, method invocation in object-oriented parlance) from the client process to web service
- ***WSDL*** – Web Services Description Language. XML vocabulary for describing the exposed public interface of the web service
- ***UDDI*** – Universal Description, Discovery, and Integration. Directory of available web services, and a runtime 'naming service' of some kind for locating the specific instance of required web service to bind to

SOAP, WSDL, and UDDI combined, define the lightweight Web Services middleware bus, and establish the basic connectivity between distributed multi-platform clients and servers on the Web. Standards are completely vendor-neutral and platform-independent, provided there is implementation of Web Services protocols for the platform in question – a safe bet, almost.

Figure 8.12 puts together artefacts that constitute the foundation of Web Services framework, and how they interoperate.

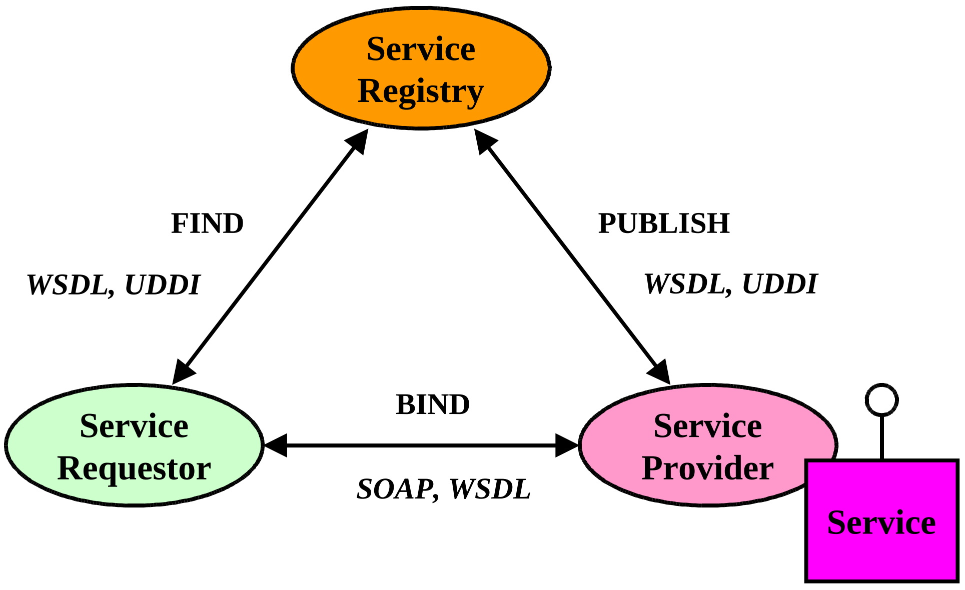


Figure 8.12. Web Services Components and Roles

Main Web Services components and their responsibilities are:

- ***Service Provider.*** Implements web service back-end infrastructure on whatever platform, and in whatever development and runtime environment it feels fit. Publishes the description of web service and its public interface in WSDL to the Registry. Keeps the web service running in order to respond to Requestor client messages
- ***Service Registry.*** Maintains directory of available web services descriptions. Allows search and querying the directory. Locates the web service for the Requestor
- ***Service Requestor.*** Finds the required web service in UDDI Service Registry. Binds to the Service Provider. Builds the defined in WSDL message, and invokes the message on web service using SOAP. Receives SOAP Response Message

Simple Object Access Protocol (SOAP) – 'come clean'

SOAP protocol is the "core of the core" of Web Services.
Web Service functionality can be implemented on any underlying platform. SOAP insulates the rest of Web Services framework from the specificity of the service implementation by defining marshalling and unmarshalling mechanism for native messages (or methods, functions, procedures, server processes) into XML format for the SOAP message, and transmitting it over Internet.

SOAP message can then become a payload on any transmission protocol, for which standard SOAP binding has been defined. First, and the most common, SOAP transport binding has been defined for the HTTP protocol as a carrier.

SOAP Request and Response messages are application-specific, and defined in terms of the application XML vocabulary in the Body of the XML SOAP Message. Ie, SOAP Body actually contains the application-specific message - method invocation with its parameters in SOAP Request, and returned value for the method in SOAP Response.
Body of the SOAP Message has to be 'wrapped' in SOAP Envelope, before the whole parcel will be shipped over the Net. SOAP Envelope, also in XML, is a generic (not application-specific, as opposed to SOAP Body) part of the transmitted XML code.

SOAP Message makes use of XML Namespaces to denote SOAP Envelope and SOAP Body, and to clarify semantics of methods and their parameters in Request and Response messages.

SOAP Request Message

```
<SOAP-ENV:Envelope
      xmlns:SOAP-ENV=
          "http://{soap-org}/{soap-envelope}"
      SOAP-ENV:encodingStyle=
          "http://{soap-org}/{soap-encoding}"
>
    <SOAP-ENV:Body>
        <m:GetStockQuota xmlns:m="some-URI">
            <symbol>ABC</symbol>
        </m:GetStockQuota>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP Response Message

```
<SOAP-ENV:Envelope
      xmlns:SOAP-ENV=
          "http://{soap-org}/{soap-envelope}"
      SOAP-ENV:encodingStyle=
          "http://{soap-org}/{soap-encoding}"
>
    <SOAP-ENV:Body>
        <m:GetStockQuotaResponse xmlns:m="some-URI">
            <sharePrice>25.87</sharePrice>
        </m:GetStockQuotaResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 8.13 SOAP Request and Response Message

Web Services Description Language (WSDL) – 'show your true face'

WSDL is similar in purpose to other Interface Definition Languages (IDL), but in the form of XML vocabulary.
WSDL describes operational information about the service – what the service does, and where the service is located.

WSDL is the convergence of separate early definition efforts of IBM and Microsoft. Currently, W3C manages the WSDL evolution, with broad industry adoption.

WSDL defines in XML two types of service description documents – *Service Interface* and *Service Implementation*.

WSDL ***Service Interface Definition*** contains WSDL elements that comprise the re-usable portion of the service description:

- ***Binding.*** Bindings describe the protocol used to access the service, as well as data formats, security, namespaces, and other attributes for the public interface of particular sevice (see *Port Type*)
- ***Port Type.*** Port Type defines *Operations* of the Web service that comprise its public interface. *Operations* define the method signature for the SOAP Message, with its Request and Response, input and output parameters (this is where WSDL and SOAP meet)

- ***Message.*** Message element specifies which XML data types constitute various parts of a message. Message element is used to define the input and output parameters of an *Operation*
- ***Type.*** Type element can be used to define complex data types within the message, typically based on XML chema

WSDL ***Service Implementation Definition*** descrtibes how a particular *Service Interface* is implemented by a given *Service Provider*:

- ***Service.*** Service element contains a collection (usually one) of *Port* elements
- ***Port.*** Port element associates the endpoint (eg. network address or URL) with *Binding* element from *Service Interface Definition*

Universal Description, Discovery and Integration (UDDI) – 'lemme find you'

UDDI is a registry, catalog or directory, and location service for the distributed Web Services, all in one.
You can think about Web Services UDDI as an amalgamation of notions from Internet DNS Bind, CORBA Naming or Trader Service, CORBA Interface and Implementation Repository, .NET Active Directory with interface repositories and location services.

Of course, you do not have to use UDDI, if you know exactly where your service is, and how to invoke it. But that would be a deterministic peer-to-peer scenario that does not work well in the dynamic business of today, and does not scale well in vast modern marketplaces on the Web.
Without UDDI, Web Sevice still can bring benefit of easy platform-independent integration between the service provider and trusted service consumers, but this connection will be tightly coupled – network connectivity, as opposed to middleware bus with intelligent service broker.

While we are at it, following this train of though, we could remove UDDI from Web Services framework, *and*, remove WSDL as well. We will be left with SOAP only, even more tightly coupled to peer-to-peer communication, and to specific implementation of the interface. Early implementations of Web Services often exhibited the cautious, conservative approach, and started small - with basic SOAP deployment.
'Bare' SOAP still can bring some benefits on it's own, e.g. by marshalling the method call into XML from some other technology, thus achieving the ease of integration. But then again, we could use pigeon carriers too, if business could tolerate latency, inflexibility, and does not have to bother about competitive advantage.
With rapidly growing maturity of Web Services standards and implementations, comprehensive use of all Web Services core components, including UDDI, will be more and more feasible.

UDDI supports three roles, or types of stakeholders in the Web Sevices – *Service Registry*, *Service Provider,* and *Service Requestor*.

Service Registry provides support for publishing and locating services.

Service Provider provides run-time e-business services, and *publishes* availability of these services through *Service Registry*.

Service Requestor finds required services by querying the Service Broker and registry, and *binds* to services with *Service Provider*.

Registry data can be maintained and provided for search via several different public directory models – *White Pages*, *Yellow Pages*, or *Green Pages* for Web Services.

White Pages let client retrieve the service details when service name is known.
Yellow Pages group businesses by business categories, applying some standard taxonomy. There are several taxonomies, and more coming – NAICS US Government industry codes, UN/SPSC (ECMA) Product and Services codes, and geographical location taxonomy.
Green Pages provide technical information about service provided by business.

UDDI allows for *private* registries, when Enterprise does not wish to publish services for public consumption, but for intranet users or partners only. *Private* UDDI may alleviate security concerns.

Secure Sockets Layer (SSL) and Transport Level Security (TLS)

The Secure Socket Layer (SSL) is a high-level security protocol that ensures privacy of the session communication between two nodes on the network. SSL is a client-server or peer-to-peer protocol. SSL is build on top of TCP/IP and transported over TCP/IP transparently to it. Being a TCP/IP payload, SSL do not have problems with Network Address Translation (NAT).
SSL currently may be used for Web, e-mail, news, and file transfer transactions.

SSL effectively adds the layer on top of TCP/IP transport, and inserts itself between the Transport and Session layers in the OSI model.

Netscape first developed SSL in 1994 as a method of protecting data sent over HTTP. Over the next two years SSL progressed to version 3.0.
In 1996, IETF formed Transport Layer Security (TLS) working group so that SSL could be adopted as an industry standard. TLS was defined in RFC2246, which was released in 1999.
TLS v1.0 is based on and largely corresponds to SSL v3.0.

On the browser, you may recognise SSL connection when URL starts with *https:* instead of *http:*. SSL connection goes through TCP port 443 to the SSL handler code on the server. Client initiates the SSL handshake with the aim of establishing the secure SSL connection or session. Initial SSL handshake phase goes over the wire unencrypted. (We use 'over the wire' as the figure of speech here, and not as an actual transmission media. Thanks to the layering of the network protocol stack, SSL cannot care less about the physical media, as long as it carries TCP).

SSL requires a reliable transport protocol TCP. SSL does not support connectionless UDP, and is not able to secure traffic that carries streaming voice and video.

SSL is a very popular due to the low impact of deployment in the existing network, and minimal or no special requirements for the client (in most cases the stock-standard Web browser will do). However, due to encryption and decryption overhead for every exchanged message, SSL may carry heavy performance price.
SSL addresses important security issues:

- Privacy and confidentiality of communication between two parties during the session using encryption
- Integrity of the exchanged messages
- Authentication of the server during the communication session. Earlier versions of SSL did not support client authentication at all

Note SSL alone does not support client authentication, although SSL v3.0 can authenticate users with X.509 Digital Certificates. You need to complement SSL with PKI and Digital Certificates for users, if strong client authentication is required. It is possible to use other than DC methods for the client authentication, like username and password, security tokens, or smart cards.

SSL nicely complements and does not overlap on OSI model with other security protocols like SOCKS, IPsec, Layer 2 Tunnelling Protocol (L2TP).

Performance of SSL may be improved by installing SSL accelerator cards on the server, as well as load balancers and content switches on the network

SSL itself is a two-layer protocol and consists of lower-level SSL Record Protocol and suite of higher-level SSL Protocols – Handshake Protocol, Change Cipher Spec Protocol, Application Data Protocol, and Alert Protocol.

Record Protocol provides basic security, encryption, segmentation, and transport functions to the upper-level SSL protocols.

Handshake Protocol allows the client and server authenticate each other, negotiate and encryption and establish SSL session connection for the private communication.
Change Cipher Spec Protocol is used to coordinate cryptographic parameters between the client and server.
Alert Protocol signals an error or warning condition to the other party in their communication.
Application Data Protocol describes the actual payload (like HTTP request) that is transmitted using the secure channel.

SSL supports following ciphers: Data Encryption Standard (DES), Triple-DES (3DES), Digital Signature Algorithm (DSA), Key Exchange Algorithm (KEA), Message Digest Algorithm 5 (MD5), Rivest Cipher 2 (RC2), Rivest Cipher 4 (RC4), Rivest-Shamir-Addleman public key algorithm (RSA), Secure Hash Algorithm (SHA-1), SKIPJACK.
RSA is the most common algorithm for the key exchange, and RC4 is the most common for data encryption. However, AES is being widely accepted as a clear direction for stronger data encryption.

TLS is an evolving standard and provides some improvements over SSL v3.
TLS v1 differs from SSL v3 in a way that keys are generated, in details of certificate verification function, alert protocol message types, and message authentication. TLS v1 application is backward compatible with SSL v3, and may downgrade to SSL v3 connection.

IETF works on enhancing and improving TLS towards better encryption, performance, and support for wireless connections.

<<< … >>>