Appendix D2

State-of-the-art technologies for REGNET

Table of Contents

1 Roles

To carry out the processes that are described in this document, the roles that will be assigned to team members is described in Table Project Roles below.

Table Project Roles

Role	Description	Partners
Co-ordinator	Overall co-ordination of whole REGNET project	AIT
Project manager	This person is the primary manager over all development resources and program management. Responsibilities include: project plan maintenance; staff allocation; user interaction; co-ordination between development.	SR
Project Control Group (PCG)	Formed by the co-ordinator (AIT) and the Project Manager (SR). These two partners are responsible for the Administrative Procedures as well as for the Technical Management of the REGNET Project.	AIT, SR
Project Management Group (nPMG)	Responsible for the overall management (co-ordinating legal and ethical issues), technical management (major decisions regarding work contents, configuration & change control, QA & self-assessment), revision of internal documentation and dissemination, and relationship with EU officers and third-party organisations.	PCG + ZEUS + VALT + IMAC + TARX + MOT + SI
Extended Project Management Group (EPMG)	This group convenes on the occasion of main project events (kick-off, milestones, contingency, etc).	PMG + all other contractors
Project Team Group (PTG)	Formed by partners working together on task level either within the development phase (I) or demonstration phase (II)	
Exploitation Committee	Formed by one management or marketing representative from the commercial partners, and will be responsible for exploitation plan and approval of dissemination activities.	
Users Group	Formed by partners acting as Content Providers or Regional Poles	
Task Leaders	Each Task will have a responsible of its specific actions (controlled by task briefs) and results	
Repository manager	The repository manager is responsible for maintaining and managing the contents and consistency of the project repository. Core to this roles activities are managing the build process.	AIT
Functional architect	BPM, UC, Object model, test scenarios, functional validation, functional aspects, vertical reuse, integration of modules developed by designers	Task leader
Technical architect	Horizontal reuse, technical validation, technical architecture definition, implementation, application, support, integration of modules developed by designers	VALT, ZEUS

Designer	Design and implementation	All technical partner

2 Risks

The aim of this part is to describe main identified risks for the REGNET system. These risks correspond to technical and functional domains as well as other possible risks (domain and management risks).

Risks are more or less probable undesirable events that will damage the project if they occur. Don't confuse risks with problems: a problem is a risk that has materialized (risk management is a proactive process, while problem management is a reactive process).

Risks are main elements for the management of a project. Risk management is an iterative process. Risks are represented by cards which can be used during the whole project.

The table below allows to determine risk criticality:

- Critical risk: Has an impact on the company and on the entire project
- Major risk: Major impact on the project
- Minor risk: Little impact

Probability of occurrence	**Severity on cost, time, performance**		
	High	Medium	Low
Very likely (P = 0.9)	Critical	Critical	Minor
Likely (P = 0.6)	Critical	Major	Minor
Unlikely (P = 0.3)	Major	Major	Minor

2.1 R1: Failure to meet deadlines

	Risk Analysis Sheet		No.
Concerning: REGNET	***Risk / Problem:*** Failure to meet deadlines		Version Date
Created by VALT	***probability*** problem (1.0) very likely (0.9) likely (0.6) unlikely (0.3)	***severity*** high medium low	***criticalness*** critical **major** minor
EFFECT with cost, timing, and justification. Impossible to deploy the first version at the end of the first year. User's evaluation and test can't be done in time. Impossible to deploy second version in time.			
CAUSE REGNET V1 system is not ready at the end of the first year. REGNET V2 is not ready at the end of the project's time.			

HISTORY / COMMENTS

Very important since delay are short and consortium very large.

N°	**Detailed actions**	**Date**	**Responsible party**	**Status/Comments**
1	Very regular monitoring of technical progress	T0	Project Manager	
2	Designation of a steering committee, with a bi-monthly project co-ordination meeting	T0	Project Manager	

2.2 R2: Over-ambitious specification

	Risk Analysis Sheet			No.
Concerning: REGNET	***Risk / Problem:*** Over-ambitious specification			Version Date
Created by VALT	***probability***		***severity***	***criticalness***
	problem	(1.0)	[✓ high]	[✓ critical]
	[✓ very likely]	(0.9)	medium	major
	likely	(0.6)	low	minor
	unlikely	(0.3)		

EFFECT with cost, timing, and justification

Development cost are more important than foreseen budget.

CAUSE

User functional requirements are very ambitious.

HISTORY / COMMENTS

Some users are not aware of the technical implication of some requirements.
User's requirements are from different domains.

N°	**Detailed actions**	**Date**	**Responsible party**	**Status/Comments**
1	A proposal to reduce functionalities		Analyst	
2	Prior functionalities will be initially implemented, others as far as it is possible.		Project Manager	

2.3 R3: Difficulty of integration

	Risk Analysis Sheet		No.
Concerning: REGNET	***Risk / Problem:*** Difficulty of integration of many software components		Version Date
Created by VALT	***probability*** problem (1.0) [✓ very likely] (0.9) likely (0.6) unlikely (0.3)	***severity*** [✓ high] medium low	***Criticalness*** [✓ Critical] Major Minor

EFFECT with cost, timing, and justification
The effect of this risk impacts on the time limit, essentially during the transitional phase.

CAUSE
REGNET is based on many software components dealing with different technologies.

HISTORY / COMMENTS
REGNET partners has different technical skill.
Some piece of software are developed in the context of an other project.
Some component already exist.

N°	Detailed actions	Date	Responsible party	Status/Comments
1	Identify the interfaces between the components		Architect	
2	Define prototypes to enable these interfaces to be validated.		Architect	

2.4 R4: Graphics interface not adapted to a type of user

	Risk Analysis Sheet			No.
Concerning: REGNET	***Risk / Problem:*** Graphics interface not adapted to a type of user			Version Date
Created by VALT	***probability***		***severity***	***Criticalness***
	problem	(1.0)	[✓ high]	[✓ Critical]
	[✓ very likely]	(0.9)	medium	Major
	likely	(0.6)	low	minor
	unlikely	(0.3)		

EFFECT with cost, timing, and justification

The effect of this risk impacts on costs, because a poor user interface can lead to poor performance for the users in their day-to-day activities, it can mean that the user interface has to be redeveloped retrospectively (which is often difficult), or it can necessitate an increased level of user training.

CAUSE

Many different users access REGNET. These users have very varied profiles (IT knowledge, language, etc.). Some of them have good knowledge of IT; others show reluctance or unease when faced with day-to-day use of computers.

HISTORY / COMMENTS

N°	Detailed actions	Date	Responsible party	Status/Comments
1	Involve an ergonomics expert.		Analyst	
2	Develop a mock-up of the interface to be validated by users with different profiles.		Architect	
3	Defined a profile based auto-adapted GUI.		Architect	

2.5 R5: Choice of new technologies

	Risk Analysis Sheet			No. R5
Concerning: REGNET	***Risk / Problem:*** Choice of new technologies			Version Date
Created by VALT	***probability*** problem very likely likely unlikely	 (1.0) (0.9) (0.6) (0.3)	***severity*** high medium low	***criticalness*** critical major minor

EFFECT with cost, timing, and justification

The effect of this risk impacts on the time limit.

CAUSE

The selected technologies (Web Services, Application servers, ebXML) has never been used by some companies of the consortium; the various employees have no development experience with it.

HISTORY / COMMENTS

N°	**Detailed actions**	**Date**	**Responsible party**	**Status/Comments**
1	Organize training.		Human Resources Management	
2	Development of prototypes or mock-ups in one of the first iterations of the development.		Architect	

2.6 R6: Rejection by users

	Risk Analysis Sheet		No.
Concerning: REGNET	***Risk / Problem:*** Rejection by users		Version Date
Created by VALT	***probability*** problem (1.0) very likely (0.9) <u>likely</u> (0.6) unlikely (0.3)	***severity*** high <u>medium</u> low	***Criticalness*** critical <u>major</u> minor

EFFECT with cost, timing, and justification
The effect of this risk impacts on the REGNET set-up period (the transitional phase) during which the users must learn to use the new system.

CAUSE
Too much sophisticated functionalities may deal with a user-unfriendly interface and unsatisfactory response times.

HISTORY / COMMENTS

N°	**Detailed actions**	**Date**	**Responsible party**	**Status/Comments**
1	Arrange interviews with the users		Analyst	
2	During deployment, train all the users		Project Manager	
3	Get certain users (to be selected) to participate in the acceptance of a prototype		Analyst	

2.7 R7: Incompatibility with some of the browsers on the market

	Risk Analysis Sheet			No.
Concerning: REGNET	***Risk / Problem:*** Incompatibility with some of the browsers on the market			Version Date
Created by VALT	***probability*** problem very likely [✓ likely] unlikely	 (1.0) (0.9) (0.6) (0.3)	***Severity*** high medium [✓ low]	***Criticalness*** Critical Major [✓ Minor]

EFFECT with cost, timing, and justification
The effect associated with this risk impacts on the quality of the product.

CAUSE
Internet browsers are not always compatible with client computers (in particular Mac). Scripting languages (JavaScript) are not always compatible with all browsers.

HISTORY / COMMENTS

N°	Detailed actions	Date	Responsible party	Status/Comments
1	Utilization of standard commands. These commands are identified in a document of coding rules.		Developer	

3 Three-tiers architectures

From here on we will only refer to 3-tier architecture, that is to say, *at least* 3-tier architecture.

A tree-tiers architecture is a software architecture where we logically and physically separate tree elements :

- The presentation tier: Is responsible for the presentation of data, receiving user events and controlling the user interface.
- The business tier: This tier isn't present in 2-tier architecture. It contain business-objects that implement the business rules, and are available to the presentation tier. This level now forms the central key to solving 2-tier problems. This tier protects the data from direct access by the clients.
- Data tier: This tier is responsible for data storage. Besides the widespread relational database systems, existing legacy systems databases are often reused here.

It is important to note that boundaries between tiers are logical. It is quite easily possible to run all three tiers on one and the same (physical) machine. The main importance is that the system is neatly structured, and that there is a well planned definition of the software boundaries between the different tiers.

Three-tiers architecture solves a number of problems that are inherent to 2-tier architectures. Naturally it also causes new problems, but these are outweighed by the advantages:

- Clear separation of user-interface-control and data presentation from application-logic. Through this separation more clients are able to have access to a wide variety of server applications. The two main advantages for client-applications are clear: quicker development through the reuse of pre-built business-logic components and a shorter test phase, because the server-components have already been tested.
- Re-definition of the storage strategy won't influence the clients. RDBMS' offer a certain independence from storage details for the clients. However, cases like changing table attributes make it necessary to adapt the client's application. In the future, even radical changes, like let's say switching form an RDBMS to an OODBS, won't influence the client. In well designed systems, the client still accesses data over a stable and well designed interface which encapsulates all the storage details.
- Business-objects and data storage should be brought as close together as possible, ideally they should be together physically on the same server. This way - especially with complex accesses - network load is eliminated. The client only receives the results of a calculation - through the business-object, of course.
- In contrast to the 2tier model, where only data is accessible to the public, business-objects can place applications-logic or "services" on the net.
- As a rule servers are "trusted" systems. Their authorization is simpler than that of thousands of "untrusted" client-PCs. Data protection and security is simpler to obtain. Therefore it makes sense to run critical business processes, that work with security sensitive data, on the server.
- Dynamic load balancing: if bottlenecks in terms of performance occur, the server process can be moved to other servers at runtime.
- Change management: of course it's easy - and faster - to exchange a component on the server than to furnish numerous PCs with new program versions. It is, however, compulsory that interfaces remain stable and that old client versions are still compatible. In addition such components require a high standard of quality control. This is because low quality components can, at worst, endanger the functions of a whole set of client applications. At best, they will still irritate the systems operator.

It is relatively simple to use wrapping techniques in 3-tier architecture. As implementation changes are transparent from the viewpoint of the object's client, a forward strategy can be developed to replace legacy system smoothly. First, define the object's interface. However, the functionality is not newly implemented but reused from an existing host application. That is, a request from a client is forwarded to a legacy system and processed and answered there. In a later phase, the old application can be replaced by a modern solution. If it is possible to leave the business object's interfaces unchanged, the client application remains unaffected. A requirement for wrapping is, however, that a procedure interface in the old application remains existent. It isn't possible for a business object to emulate a terminal. It is also important for the project planner to be aware that the implementation of wrapping objects can be very complex.

4 Design pattern

4.1 Observer (behavioural pattern)

Problem: Sometimes many objects, called *observers*, will be interested in significant events related to one object, called the *subject*. For instance, *observers* are often interested in changes in the *subject's* state. A classic example of this relationship is a group of windows displaying different views of the same information. Each window has a vested interest in changes in the information contained in the *subject* document. When a change occurs, a notification message must be sent from a *subject* to each of its *observers.*

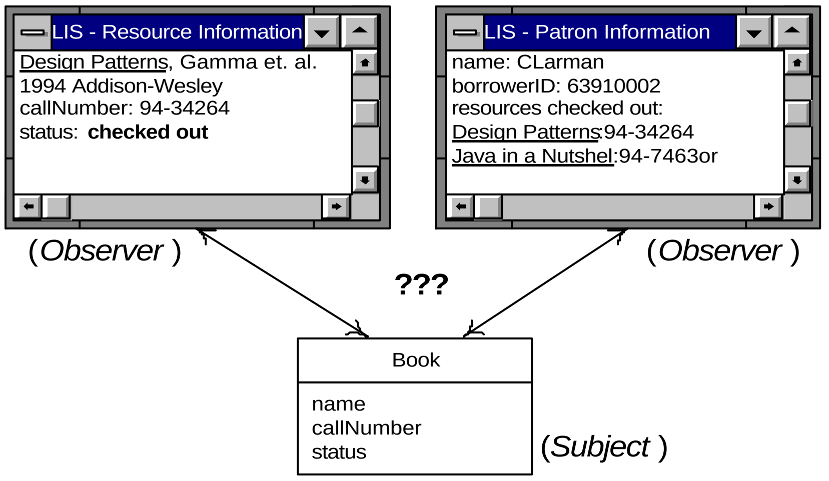


Figure 1 Connection subject - observer

Why is it undesirable to directly couple the *subject* to its *observers*? There might be several reasons:

- In many systems, the number of *observers* may be dynamic (e.g. in the above example, windows are created and destroyed).
- The types of *observers* may vary. In other words, there may be a "mixed bag" of *observers* (e.g. different kinds of windows viewing the same information).
- The whole set of *observers* might change (e.g. the GUI is changed entirely). This is related to the multi-tiered architecture.

Solution: We construct a notification framework. This adds a level of indirection that lessens the coupling between *subjects* and their *observers*. This is accomplished by creating Subject and Observer classes that are subclasses by your specific domain objects.

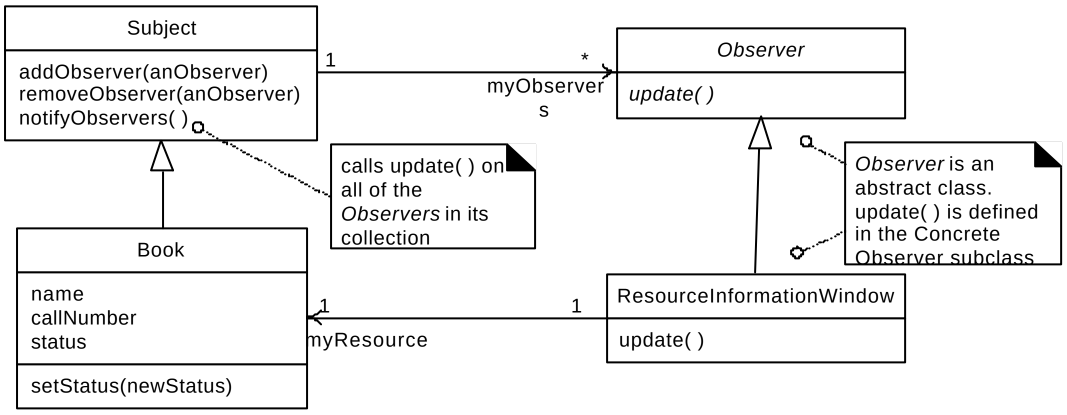

Subject
addObserver(anObserver)
removeObserver(anObserver)
notifyObservers( )
1
*
myObservers
Observer
update( )
calls update( ) on all of the Observers in its collection
Observer is an abstract class. update( ) is defined in the Concrete Observer subclass
Book
name
callNumber
status
setStatus(newStatus)
1
myResource
1
ResourceInformationWindow
update( )

Figure 2 Subject and observer classes

In this example, Book (the concrete Subject) inherits from Subject. The superclass contains all of the functionality associated with maintaining a list of Observers. It is also responsible for sending an update() message to each Observer when notifyObservers() is called from the subclass. Since setStatus() is a method that changes the state of Book, it is a likely location for the notifyObservers() method to be invoked. On the other side of the framework, ResourceInformationWindow is the Concrete Observer that subclasses Observer. The Concrete Observer defines the actual functionality of the update() method. Since update() will probably need access to information in the Concrete Subject, a Concrete Observer generally has visibility to its Concrete Subject.

This is the simplest design of Observer described in the Design Patterns book. It is now considered inadequate for most systems. Observer is now usually implemented using a more sophisticated event framework. One example of this is the Java AWT event model.

The advantage Observer is that domain objects are less strongly coupled to their Observers. They are also encapsulated from the details of notification and are required only to make the generic call, notifyObservers(). Additionally, since any class can subclass Observer, Concrete Observers may be of any type.

Related Patterns: The Mediator pattern defines an object that manages the communication between objects in a many to many relationship. A Mediator could be used in a more sophisticated design of the Observer pattern.

5 J2EE Architecture

Application servers are the answer to three main objectives: propose a development environment, support integration of legacy application and facilitate application deployment. Java adds the simplicity and standardisation.

Three-tiers architectures has been defined in order to solve these problems, but implementation complexity and lack of tools prevent, until recent time, their adoption. Application server gives usable solutions by providing technical framework.

Application server utilisation provides development simplification: application development on such a framework consist in developing components and to deposit them into the server. In such way, development team writes little technical code, the application server gives technical framework in order to integrate components.

5.1 Description

E-Business applications are moving to open and standards-based technologies. The trend is encouraging a growing number of Web applications package vendors to embrace the J2EE specification. The specification is the result of an industry partnership and open community process led by Sun Microsystems.

The J2EE Platform provides a component-based approach to the design, development, assembly, and deployment of enterprise applications. The J2EE platform is designed to provide server-side and client-side support for developing enterprise, multi-tier applications. Such applications are typically configured as a client tier to provide the user interface, one or more middle-tier modules that provide client services and business logic for an application, and backend enterprise information systems providing data management.

The J2EE platform provides a multi-tier distributed application model. This means that the various parts of an application can run on different devices. The J2EE architecture defines a client tier, a middle tier (consisting of one or more sub-tiers), and a backend tier providing services of existing information systems. The middle tier supports client services through Web containers in the Web tier and supports business logic component services through Enterprise JavaBeans containers in the EJB tier. The enterprise information system (EIS) tier supports access to existing information systems by means of standard APIs.

Central to the J2EE component-based development model is the notion of containers. Containers are standardized runtime environments that provide specific components services. Components can expect these services to be available on any J2EE platform from any vendor (e.g. transaction, EJB life cycle management). Containers also provide standardized access to enterprise information systems; for example, providing RDBMS access through the JDBC API. In addition, containers provide a mechanism for selecting application behaviours at assembly or deployment time.

With a set of features designed specifically to expedite the process of distributed application development, the J2EE platform offers several benefits:

- Simplified architecture and development
- Scalability to meet demand variations
- Integration with existing information systems
- Choices of servers, tools, components
- Flexible security model

Enterprise application developers use the services through a set of Java technologies or APIs mandated by the J2EE specification. Because developers do not have to worry about low-level service details, it is easy to develop multi-tiered, distributed, scalable Java applications. J2EE services also provide for customizing applications at deployment.

J2EE application developers write application components. An application component is a self-contained module that is added to and interfaces with other application components. Application components include thin-client applications, applets, servlets, Java Server Pages (JSPs), and server-side Enterprise JavaBeans (EJBs).

A typical J2EE application comprises presentation, business logic, and data tiers.

J2EE architecture for a typical J2EE application is based on three tiers.

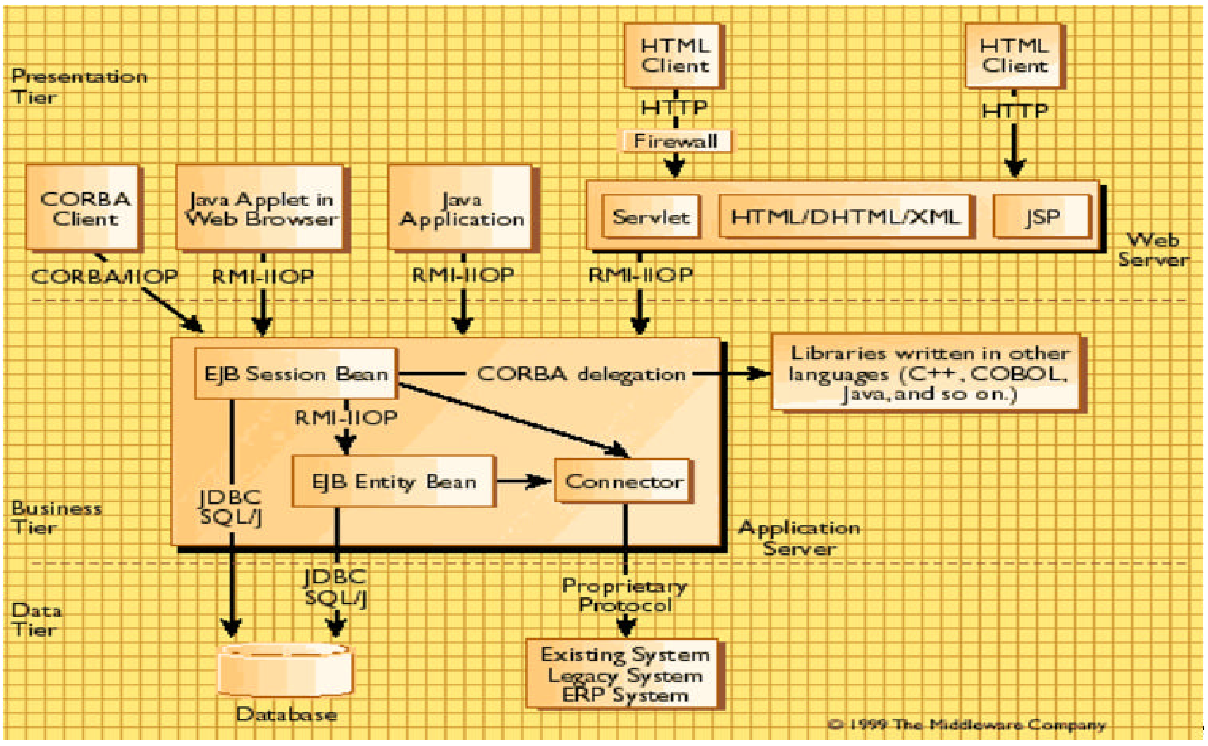


Figure 3 three-tier architecture

The presentation tier clients can be CORBA clients, applets, Java applications, and JSPs or servlets. The business tier contains a mix of Session (process) and Entity (data) EJBs. The EJBs run inside containers provided by an application server. The EJBs rely on their containers and the application server to provide transaction, state management, persistence, security, and resource pooling services. Any client component needing the services of an EJB locates its reference through JNDI and uses RMI-IIOP for method invocation. Figure shows that the JDBC API is used to integrate with relational or tabular data sources, and connectors are used to integrate with non-relational data sources and EISs.

Main components of the J2EE architecture are described below: JSP, Servlet, EJB, JDBC.

5.2 Presentation tier: JSP, Servlet

JavaServer Pages (JSP) offers a 100% Pure Java alternative to Microsoft's proprietary Active Server Pages (ASP). JSP technology extends Java servlet technology, and, in fact, the JSP framework translates JSP pages into servlets at run time. Servlets are popular because they supply architectural and performance advantages over CGI scripts. Servlets can also generate dynamic Web pages by mixing static HTML with content supplied by database queries or business services. JavaServer Pages invert this approach by imbedding Java code in HTML. This ability to insert Java code into HTML pages adds flexibility to servlet-based Web architectures.

To generate HTML, servlets must supply formatted strings to println() calls. This technique clogs Java code with line after line of hard-to-comprehend HTML. Further, when servlets generate HTML, Web page design requires programmers. JavaServer Pages pull HTML out of Java code and create a role for HTML designers. Site development can proceed along parallel tracks—Java design and HTML design—thereby delivering a Web site faster. JavaServer Pages also encourage loose coupling between business logic components and presentation components, thereby making reuse of both more likely.

5.3 Business Tier: EJB

The Enterprise JavaBeans (EJB) architecture is a server-side technology for developing and deploying components containing the business logic of an enterprise application. Enterprise JavaBeans components are scalable, transactional and multi-user secure.

Enterprise beans are hosted by an EJB container. In addition to standard container services, an EJB container provides a range of transaction and persistence services and access to the J2EE service and communication APIs.

There are two types of enterprise beans: session beans and entity beans.

Session Beans

A session bean is created to provide some service on behalf of a client and usually exists only for the duration of a single client-server session. A session bean performs operations such as calculations or accessing a database for the client. While a session bean may be transactional, it is not recoverable should its container crash.

Session beans can be stateless or can maintain conversational state across method and transactions. If they do maintain state, the EJB container manages this state if the object must be removed from memory. However, the session bean object itself must manage its own persistent data.

Entity Beans

An entity bean is a persistent object that represents data maintained in a data store; its focus is data-centric. An entity bean can manage its own persistence or it can delegate this function to its container. An entity bean can live as long as the data it represents.

An entity bean is identified by a primary key. If the container in which an entity bean is hosted crashes, the entity bean, its primary key, and any remote references survive the crash.

5.4 Data tier: JDBC

JDBC is a vendor-independent API for accessing relational data sources. It is easy to use JDBC to send SQL statements to virtually any relational database for retrieving and updating data (Oracle, Sybase, DB2, Informix, and SQLServer, to name some of the most pervasive examples). First specified in 1997, the JDBC API initially focused on basic call-level interfaces to SQL databases. JDBC 2.1 and Optional Package 2.0 extended the API's capabilities to collaborate with J2EE application servers, providing connection management and distributed transactional support across multiple databases.

JDBC's benefits are:

- Developers program to a common client API based on SQL. A developer who knows the JDBC API can program for any relational database. Relational database vendors do not have to provide different drivers for different application servers. If a driver is JDBC-compliant (JDBC 2.1 and the 2.0 Optional Package), the driver vendor is assured that its driver will work with any J2EE-compliant application server.
- Application server vendors do not have to write special code to accommodate different drivers. They extend their application server once to comply with J2EE and JDBC requirements and open themselves to interoperability with a wide range of JDBC drivers.

5.5 J2EE application servers

Developing J2EE applications requires a J2EE-compliant application server. The application servers come with an HTTP server, a database, and deployment tools. Because the servers comply with J2EE, they also provide support for additional services, including naming and access, resource pooling, transactions and security.

J2EE specification is currently release to version 1.2 but the majority of the available tools are currently in version 1.1.

There are plenty of industrial tool both commercial or open source. A detail matrix of all the J2EE application server is available from flashline at: http://www.flashline.com/components/appservermatrix.jsp.

5.6 J2EE resources

There are plenty of J2EE resources available on the Net. More relevant are:

Sun page: http://java.sun.com/j2ee/

http://www.theserverside.com/home/index.jsp

J2EE blueprints from Sun: http://java.sun.com/j2ee/blueprints/index.html

Texmetrix: http://www.techmetrix.com/

6 PHP

6.1 Introduction

PHP is short for what is officially called "PHP: Hypertext Preprocessor." PHP was created by Rasmus Lerdorf, a developer who needed a tool to keep track of who was looking at his resume. The first version of Lerdorf's preprocessor (known then as PHP/FI) was released in 1994. Since then, PHP has become popular in the open-source community and was renamed PHP with the release of version 3.0. PHP is now at version 4.0 and is used by over 3 million web sites on the Internet.

PHP represents a robust open-source development language that provides the tools and fexibility to accomplish virtually any task. PHP is an embedded language which means developers can jump between raw HTML code and PHP without sacrificing readability. Beyond its basic syntax, PHP also boasts a wide range of interfaces allowing it to communicate with everything from other web pages, to databases including ODBC, and other programming languages such as Java or COM.

For those who are familiar with web development, the term CGI (Common Gateway Interface) may be familiar to them. CGI provides a way for developers to write computer programs that can construct HTML and process data from web pages dynamically. Before CGI, web developers were forced to write static HTML pages that required tedious manual updates.

PHP is a customised, embedded CGI language. Because PHP is server-side technology, the person viewing the web page needs no special programs or browser plug-ins for PHP to work. PHP is compatible with all major web browsers, and although it is classified as a CGI, PHP is a tool that provides much more power. It allows a web developer to dynamically construct a web page based on data gathered from a third source (a database or otherwise) and then communicate that data through almost any means provided by the Internet. The real benefit of this is the developer can do these things with little or no knowledge of the inner workings between the CGI and the database they are communicating with.

As mentioned before PHP is a hypertext preprocessor. In a less technical sense, this means that when a user points a browser at your web site, PHP gets a chance to make "last-minute" changes to the page before the user sees it (See next Figure).

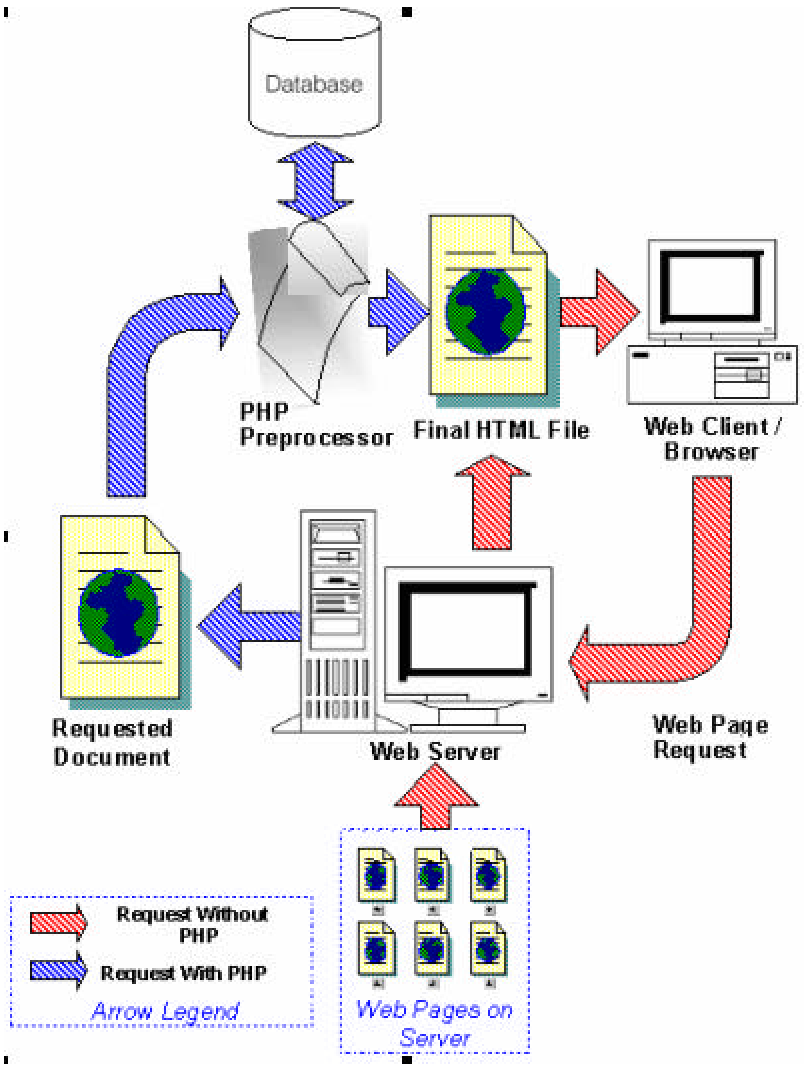


Figure 4 Web request processing with and without PHP.

6.2 PHP Syntax

PHP's basic syntax is familiar.

```
<?php
echo "Hello, World!";
?>
```

produces

```
Hello, World!
```

Variables are marked with a preceding $. The sentence "Hello, World!" can be printed on the screen like this:

```
<?php
$message = "Hello, World!";
echo $message;
?>
```

String concatenation is done with . (a period); other arithmetic operators are:

```
<?php
$greeting = "Hello ";
$num = 3 + 2;
$num++;
echo "$greeting $num people!";
?>
```

produces

```
Hello 6 people!
```

PHP has the full complement of operators, and they work just as it was expected to--especially for the persons with a background of C or C++. A good rule of thumb for PHP: "When in doubt, try it; it will probably work."

Just as in Perl, a string surrounded with double quotes causes variables inside it to be interpolated, but a string surrounded with single quotes does not. So,

```
<?php
$name = 'Susannah';
$greeting_1 = "Hello, $name!";
$greeting_2 = 'Hello, $name!';
echo "$greeting_1\n";
echo "$greeting_2\n";
?>
```

produces

```
Hello, Susannah!
Hello, $name!
```

Note that the \n in the string turns into a new line, just as in Perl or in C. This only works in double-quoted strings, however.

Variables

PHP makes environment variables available to you as regular variables. This includes the environment variables that are set by the server for a CGI program (even if you're running PHP as a module). For this reason, if the page "http://www.domain.com/farm/cattle/cow-cow.cow.html" contains the code

```
<?php
echo "[$REQUEST_URI]";
?>
```

it prints out [/farm/cattle/cow-cow-cow.html]

Arrays

It is possible to set off array indices (regular or associative) with square brackets ([and]):

```
$fruit[0] = 'banana';
$fruit[1] = 'papaya';
$favorites['animal'] = 'turtle';
$favorites['monster'] = 'cookie';
```

If something is assigned to an array but the index is left blank, PHP assigns the object onto the end of the array. The statements about $fruit, above, produce the same result as:

```
$fruit[] = 'banana';
$fruit[] = 'papaya';
```

The multidimensional arrays are being generated as follows:

```
$people['David']['shirt'] = 'blue';
$people['David']['car'] = 'minivan';
$people['Adam']['shirt'] = 'white';
$people['Adam']['car'] = 'sedan';
```

A shortcut for creating arrays is the

array()

function:

```
$fruit = array('banana','papaya');
$favorites = array('animal' => 'turtle',
                   'monster' => 'cookie);
```

or

```
$people = array ('David' => array('shirt' => 'blue',
                                  'car' => 'minivan'),
                 'Adam' => array('shirt' => 'white',
                                  'car' => 'sedan'));
```

The built-in function count() informs on how many elements are in an array:

```
$fruit = array('banana','papaya');
print count($fruit);
```

prints

```
2
```

Control Structures
The looping structures are being used with the for and while:

```
for ($i = 4; $i < 8; $i++) {
  print "I have eaten $i bagels today.\n"; }
```

prints

```
I have eaten 4 bagels today.
I have eaten 5 bagels today.
I have eaten 6 bagels today.
I have eaten 7 bagels today.
```

So does

```
$i = 4; while ($i < 8) {
  print "I have eaten $i bagels today.\n";
  $i++;
}
```

It is possible to use the control structures if and elseif:

```
if ($user_count > 200) {
   print "The site is busy right now!";
} elseif ($user_count > 100) {
   print "The site is sort of active right now!";
else {
   print "The site is lonely - only $user_count user logged on.";
}
```

The rule of thumb about operators also applies to control structures. It is possible to use switch, do...while, and even the ?: construct.

6.3 Object Oriented Programming (OOP) with PHP

The high-paced development schedules of today's software has only punctuated the need for developers to possess efficient software strategies. One such strategy, known as *object-oriented programming* (OOP), has stepped to the forefront as the de facto standard for implementing strong software design. I'll examine the basic concepts of OOP and gain practical insight into how PHP can be used to develop OOP applications.

PHP, a popular server-side Web scripting language, is a logical tool for those wishing to keep up to pace with the application development feld. Even novice programmers can build fantastically dynamic and flexible applications. Coupling PHP with an object-oriented strategy serves to optimise time and lets developers efficiently create cutting-edge applications.

Before delving into a series of examples illustrating just how this can be implemented, let's begin with a short introduction to OOP.

6.3.1 Practical OOP

OOP emphasizes the state, behaviour, and interaction of data within an application. By focusing upon the data, the developer is better able to model the environment which he or she is attempting to implement. OOP results in more readable code, and greatly enhances the flexibility and management of a system.

One of the main advantages of OOP is that it enables programmers to build customised data modules, each having specific attributes and functionality. In OOP lingo, this customised data module is known as a *class.*

A class enables the programmer to model an application code upon real world concepts. Essentially, it can be thought of as an extended data type which goes beyond the traditional mission of a data type, defining not only attributes, but also functionality specific to that data type. Much as a variable is assigned the attributes of a data type, a new type of variable, known as an *object,* can be assigned the attributes and methods declared within a class.

To put it simply, a class can be thought of as a cookie-cutter. The class provides a template from which one or more objects can be made.

6.3.2 Building classes

Defining classes in PHP is a rather straightforward task. Many of the programming concepts involved with class declaration are alike to traditional object oriented programming languages.

6.3.3 Building Objects

In PHP, as with most other object-oriented languages, an object is declared using the new keyword. This keyword specifies that a new object is to be created based upon the class reference following the new keyword. In order to understand better the above a short example follows:

```
<?
// example 1
// . . . Vehicle class declaration
// declare a few objects
$motorcycle = new Vehicle;
$automobile = new Vehicle;
$plane = new Vehicle;
?>
```

Now that the objects have been created, an interaction with them can begin. This is where the advantages of OOP begin to shine through, particularly when it comes down to code readability and convenience.

6.3.4 Using Objects

A created object has its own copy of the attributes, and has use of all methods declared within that class. This is illustrated in the following example: Now that the objects have been created, the modification of their attributes is possible

```
<?
// example 2

$motorcycle->set_model("Harley");
$motorcycle->go(55);

$automobile->set_model("Corvette");
$automobile->go(95);

$model = "Cessna 172";

$plane->set_model($model);
$plane->go(165);
?>
```

Executing this code along with the above examples, the following would be output to the screen:

```
Vehicle model set to Harley.
The Harley is going 55 m.p.h..
Vehicle model set to Corvette.
The Corvette is going 95 m.p.h..
Vehicle model set to Cessna 172.
The Cessna 172 is going 165 m.p.h..
```

To summarize, a declaration of three new objects ($motorcycle, $automobile, and $plane) has been established. The attributes of these newly created objects were then modified using the methods found within the Vehicle class.

6.3.5 Inheritance

Judging from the simple examples above, the idea of basing the code on objects is certainly a potent one. However, this is only the basis for more powerful concepts of OOP. Not only is possible to develop these customised data modules, but the developer is free to build relationships between these modules, leading to a highly structured and flexible programming environment.

One of the premises for building these relationships is known as *inheritance.*

Much like a child inherits characteristics from his or her parent, derived classes inherit the characteristics of the class in which they are derived, and are also known as *child* and *parent* classes, respectively.

So, suppose the developer needs to further divide the Vehicle class into three subclasses: land, air, and water. All three of these subclasses should share the general attributes and functions of the Vehicle class, but also have attributes which are useful only within each specific subclass (wingspan for the air subclass, for example). To eliminate the unnecessary redeclaration of these attributes within each subclass, the subclass instead inherits the characteristics by a concept known as, conveniently enough, inheritance.

Another reason to use inheritance would be to further organize data structures into strictly meaningful entities. This means that only the information that is relevant to that category will be included, and nothing else. After all, what use would wingspan have when implementing the water or land subclass? On the same note, num_wheels would have no place within the water subclass.

Inheritance with PHP:

The base class (Vehicle) will serve as the parent for the three child classes: land, air, and water. As stated, it is desired the three children to share the same attributes as possessed by the Vehicle class, in addition to owning their own characteristics specific to each individual purpose.

Putting the newly defined subclasses to use should clearly illustrate the power of inheritance:

```
<?
$my_jetski = new Water_vehicle;
$my_jetski->set_model("Sea-Doo");
$my_jetski->set_anchor(1);
?>
```

The resulting output:

```
Vehicle set to Sea-Doo.
Anchor is up.
```

This example not only illustrates the usage of the methods within the Water_vehicle subclass, but also those of the parent (Vehicle).

6.4 Summary

PHP is a great tool for quickly bringing dynamic elements to the Web site. With its familiar syntax and low overhead, getting started is easy – and there is no worry about being bogged down in details when all it is needed is a simple script.

What's in this chapter is just the beginning of PHP's power. PHP lets the user to easily store and retrieve information from a gamut of databases: MySQL, Oracle, Microsoft SQL Server, and more. The creation and modification of images is being held on the fly. The feature of text manipulation with regular expressions is being provided by PHP.

In addition to the previous even the object oriented programming is being solved by the unlimited powers of PHP and it is easy to understand that PHP is the total solution for the World Wide Web and the relevant technology issues are being confronted in a magnificent manner by the PHP abilities.

7 Web services

This appendix gives information about technologies used in order to integrate REGNET software modules based on Web services approach.

Web Services allow enterprises to implement business process and trading partner agreements over the Internet, using XML messages for input and output. Web Services software defines a framework to publish and interact with Internet *services* that access application programs directly, without browsers or HTML files. Applications talk directly to each other and exchange business data directly, without browsers. Web Services can also provide access to standard library functions such as security, transactions, translation, search, credit card validation, catalogue management, logging, and so on, and entire enterprise applications can be assembled from them.

The technologies required include:

- XML, including basic XML, XML schemas, XML parsers, and XML transformation tools.
- •Simple Object Access Protocol (SOAP), a specific usage of XML as an application-to-application protocol, including *an optional* serialization format and RPC-style mapping.
- Web Services Description Language (WSDL), a specific form of an XML Schema used to describe Web service messages, operations, and protocol mappings.

- Universal Description, Discovery, and Integration (UDDI), a repository for registering and discovering Web service descriptions.
- •ebXML, RosettaNet, and other business process standards for document types, trade partner interactions, and industry-specific XML vocabularies.

7.1 SOAP

The Simple Object Access Protocol (SOAP) is an XML-based messaging framework specifically designed for exchanging formatted data across the Internet (for example using request and reply messages or sending entire documents). SOAP is simple, easy to use, and completely neutral with respect to operating system, programming language, or distributed computing platform.

As well as providing a mapping to a transport layer for exchanging XML messages across the Internet, SOAP also supports a framework within which a company can:

- publish its services for exchanging XML business data;
- discover the location and format of other business services
- specify quality of service attributes for the messages (see sections on WSDL and UDDI for further information on other aspects of the framework).

SOAP provides an independent, abstract communication protocol capable of bridging or connecting two or more business's enterprise portals, or two or more remote business sites. The enterprise portals can be built using any combination of hardware and software that supports Internet access to existing systems. The existing systems typically represent multiple different infrastructures and packaged software products. SOAP and XML provide the means for any two or more portals, marketplaces, or trading partners to agree on common data exchange services for exposing services to the Web on behalf of the integrated existing systems. SOAP, XML, and WSDL can also be used to integrate the existing systems.

Basic steps involved in discovering data services published to the Internet by an enterprise's portal:

- Enterprise A uses the URL provided by Enterprise B to obtain a list of published services.
- Enterprise A downloads the XML schemas (usually WSDL) describing the format(s) of the messages the services expect.
- Enterprise A formats an XML message accordingly and sends it via SOAP to Enterprise B.
- Enterprise B returns the response via SOAP, which Enterprise A also interprets using the XML schema information.

In this way, the two enterprises can use existing Internet infrastructure to exchange information about the services that they wish to publish and consume. For example, Enterprise B might want to publish its message format for a purchase order for bulk discount orders on large screen televisions. Enterprise A, having obtained the Internet address (that is, the URL) of Enterprise B's service description from Enterprise B's Web site, in an email, or in a print advertisement, includes Enterprise B's URL in a list of URLs that it is checking for favorable pricing and delivery terms. When Enterprise A has discovered and exchanged information with its list of suppliers, it can send the actual purchase order to the supplier with the most favorable terms and conditions.

Using SOAP directly in a portal, for example, Enterprise A might send a formatted XML purchase order document to Enterprise B. Enterprise B's portal would decode the purchase order and route the information to one or more existing order fulfillment systems for processing. After the order was filled and shipped, Enterprise B would send a formatted XML document back to Enterprise A, using SOAP.

Another method of publishing and discovering Web service definitions is the

Universal Description, Discovery, and Integration (UDDI) registry (Universal Description, Discovery, and Integration (UDDI) for more details).

The current SOAP specification (V1.1) is accessible from the World Wide Web Consortium (W3C) Web site (www.w3.org/TR/SOAP). Eleven companies, submitted SOAP to the W3C, which established a working group on XML protocols to evaluate SOAP and produce an industry recommendation for XML Protocol use. In general, SOAP messages must be viewed together with their associated XML schemas, which tell the consumers of SOAP messages how to understand them.

SOAP is currently under development, and details change. However, the current version of the specification supports two basic XML message formats:

- •Self-describing for EAI and EDI style document exchanges.
- •Remote procedure call (RPC) style interactions that model object method invocation and parameter passing.

In both cases SOAP uses associated XML Schemas to define the message and its type(s). The XML Schema and type information used in SOAP is qualified using XML namespaces; another XML extension provides XML element name scope (that is, the element names are uniquely qualified using the XML name space extension).

SOAP V1.1 describes how the data types defined in associated XML schemas can be serialized, or marshaled, over HTTP (or other) transports. Both the publisher and consumer of SOAP messages must have access to the same XML schemas in order to correctly exchange information. The schemas are normally posted on the Internet, and may have to be downloaded by any one or more than one party in an exchange of messages.

SOAP messages consist of three primary parts:

- •Envelope
- •Header (optional)
- •Body

The envelope is required, and basically marks the start and end of the SOAP message (although messages can contain links to objects outside of the envelope).

The header is optional, and can be used to include information about the RPC style interaction, if needed, or to carry other general information about the message, such as security information, mail-to addresses for the message (if any), or payment codes for purchase orders. A SOAP message can include none or any number of headers.

The body carries the data for the actual message or document being sent. Optional encoding rules specify the format of the data on the wire for both the header and the body. SOAP message elements are defined using schemas and qualified using XML namespaces.

7.2 WSDL

After SOAP became available as a mechanism for exchanging XML messages among enterprises (or among disparate applications within the same enterprise), a better way was needed to describe the messages and how they are exchanged. The Web Services Description Language (WSDL) is a particular form of an XML Schema, developed by Microsoft and IBM for the purpose of defining the XML message, operation, and protocol mapping of a Web service accessed using SOAP or other XML protocol. WSDL defines Web Services in terms of "endpoints" that operate on XML messages. The WSDL syntax allows both the messages and the operations on the messages to be defined abstractly, so they can be mapped to multiple physical implementations.

The current WSDL spec describes how to map messages and operations to SOAP 1.1, HTTP GET/POST, and MIME. WSDL creates Web service definitions by mapping a group of endpoints into a logical sequence of operations on XML messages. The same XML message can be mapped to multiple operations (or services) and bound to one or more communication protocols (using "ports"). A WSDL schema includes the following parts:

- •**Type**s—define data types to be used in the messages, in the form of XML
- Schemas or possibly other mechanisms.
- •**Messag**e—an abstract definition of the data itself in the form of a message

- (presented either as an entire document or in the form of arguments to be
- mapped to a method invocation).
- •**Operatio**n—abstract definition of the operation for a message, such as naming
- a method, message queue, or business process that will accept and process
- the message.
- •**Port Typ**e—an abstract set of operations mapped to one or more endpoints,
- defining the collection of operations for a binding (the collection of operations,
- since it's abstract, can be mapped to multiple transports through different
- bindings).
- •**Bindin**g—a protocol and data format for a port type that defines how the port
- type is mapped onto a particular transport.
- •**Por**t—a combination of a binding and a network address, providing the target
- address of the service communication.
- •**Servic**e—a collection of related endpoints encompassing the service definitions in the file—services map the binding to the port and include any extensibility definitions.

WSDL is a collection or list of definitions that comprises three main parts:

- A definition of message types and operations in the abstract.
- A definition that maps the abstract definitions onto concrete transports and network end points.
- A definition that maps bindings to ports and includes extensibility (attributes or properties for quality of service agreements for example).

The "binding mechanism" is used to map a particular XML message to a specific protocol or data format. Like SOAP, WSDL uses XML namespaces to ensure the uniqueness of element names. Often, the WSDL file will be generated from another type of information, such as a CORBA or COM IDL file or EJB class definition.
An example for a WSDL document is given below:

7.3 UDDI

The Universal Description, Discovery, and Integration (UDDI) framework defines a data model (in XML) and SOAP APIs for registration and searches on business information, including the Web Services a business exposes to the Internet. UDDI is an independent consortium of vendors, founded by Microsoft, IBM, and Ariba, for the purpose of developing an Internet standard for Web service description registration and discovery. Microsoft, IBM, and Ariba also are hosting the initial deployment of a UDDI service, which is *conceptually* patterned after DNS (the Internet service that translates URLs into TCP addresses).

UDDI uses a private agreement profile of SOAP (UDDI does not use the SOAP serialization format because it is not well suited to passing complete XML documents (it is aimed at RPC style interactions). The main idea is that businesses use the SOAP APIs to register themselves with UDDI, and other businesses search UDDI when they want to discover a trading partner (for example, someone from whom they wish to procure sheet metal, bolts, or transistors). The information in UDDI is categorized according to industry type and geographical location, allowing UDDI consumers to search through lists of potentially matching businesses to find the specific one they want to contact. Once a specific business is chosen, another call to UDDI is made to obtain the specific contact information for that business.

The contact information includes a pointer to the target business's WSDL or other XML schema file describing the Web service that the target business publishes.

7.4 Tools

PHP/Java integration.

8 E-Business

In the e-Business sector a new standard is available that provides an open XML-based infrastructure enabling the global use of electronic business information in an interoperable, secure and consistent manner by all parties.

ebXML (www.ebxml.org), sponsored by UN/CEFACT and OASIS, is a modular suite of specifications that enables enterprises of any size and in any geographical location to conduct business over the Internet. Using ebXML, companies now have a standard method to exchange business messages, conduct trading relationships, communicate data in common terms and define and register business processes.

- **ebXML Value**
  - Provides the only globally developed open XML-based Standard built on a rich heritage of electronic business experience.
  - Creates a Single Global Electronic Market Enables all parties irrespective of size to engage in Internet-based electronic business. Provides for plug and play shrink-wrapped solutions.
  - Enables parties to complement and extend current EC/EDI investment expand electronic business to new and existing trading partners.
  - Facilitates convergence of current and emerging XML efforts.
- **ebXML delivers the value by**
  - Using the strengths of OASIS and UN/CEFACT to ensure a global open process.
  - Developing technical specifications for the open ebXML infrastructure.
  - Creating the technical specifications with the world's best experts.
  - Collaborating with other initiatives and standards development organizations.
  - Building on the experience and strengths of existing EDI knowledge.
  - Enlisting industry leaders to participate and adopt ebXML infrastructure.
  - Realizing the commitment by ebXML participants to implement the ebXML technical specifications.

8.1 General framework

The basic concept of the e-Business subnode will be based on the ebXML specifications. Therefore our approach will establish the ebXML configuration of the e-Business subsystem, which will reflect the special needs of the REGNET project.

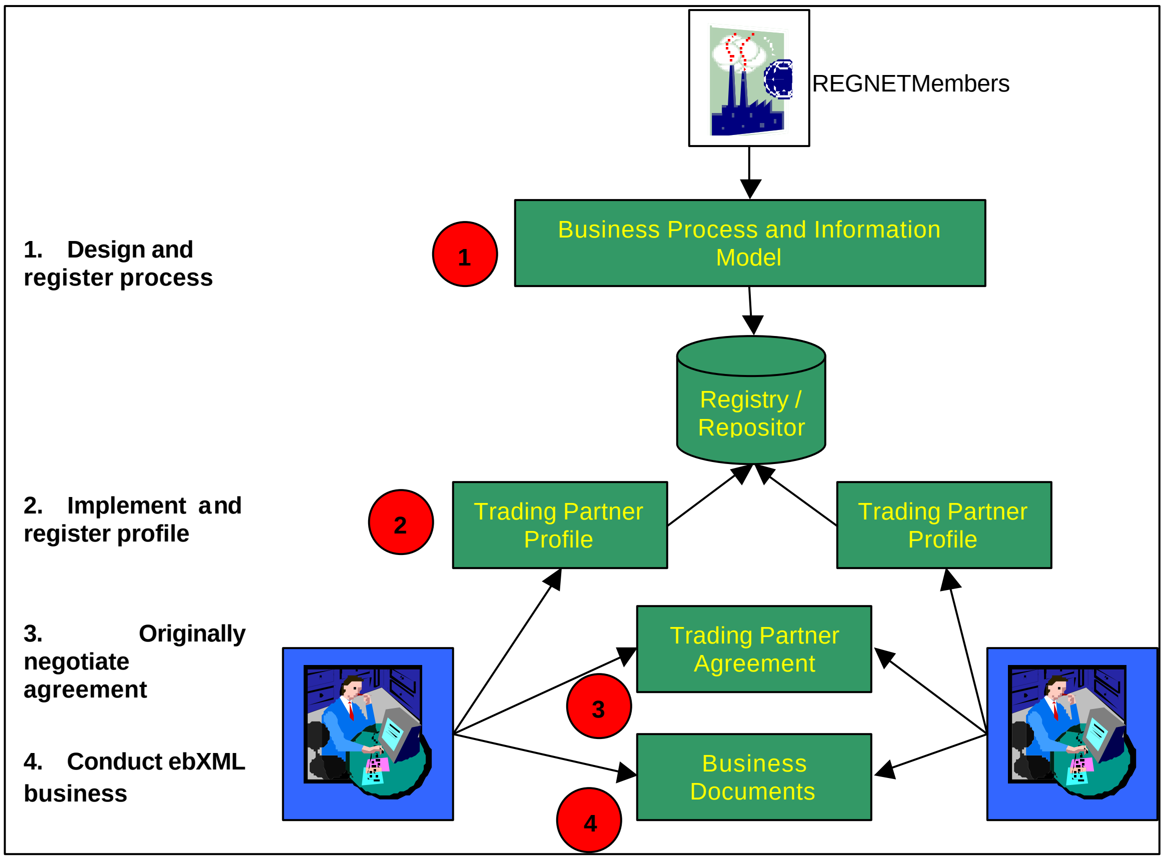

REGNETMembers
1. Design and register process
1
Business Process and Information Model
Registry / Repositor
2. Implement and register profile
2
Trading Partner Profile
Trading Partner Profile
3. Originally negotiate agreement
Trading Partner Agreement
3
4. Conduct ebXML business
Business Documents
4

Figure 5 e-Business general framework

The above figure shows the 4 necessary steps that need to be implemented, in order to support the ebXML specifications in the e-Business subsystem. Therefore the design and the functional requirements of the e-Business subsystem are the following:

1. Design and register business processes and information models
   - The implementer browses the repository for appropriate business processes, or for the process the intended partner is registered to support
2. Implement business service interfaces and register Collaborative Partner Profiles
   - The implementer buys, builds, or configures application) capable of participating in the selected business process.
   - The implementer registers his capability to participate, in the form of a Collaborative Partner Profile.
3. Optionally negotiate and define a Collaborative Partner Agreement (CPA)
   - The two parties negotiate technical details and/or functional overrides, and draw up the result in the form of a CPA
4. Parties optionally register the CPA

The above steps are necessary to support the eBusiness subsystem, which is described by the ebXML specifications.

9 ebXML Infrastructure

9.1 Trading Partner Information

9.1.1 Introduction

To facilitate the process of conducting *e-Business*, potential *Trading Partners* need a mechanism to publish information about the *Business Processes* they support along with specific technology implementation details about their capabilities for exchanging business information. This is accomplished through the use of a *Collaboration Protocol Profile (CPP).* The *CPP* is a document which allows a *Trading Partner* to express their supported *Business Processes* and *Business Service Interface* requirements in a manner where they can be universally understood by other ebXML compliant *Trading Partners*.

A special business agreement called a *CPA* is derived from the intersection of two or more *CPP's*. The *CPA* serves as a formal handshake between two or more *Trading Partners* wishing to conduct business transactions using ebXML.

9.1.2 CPP Formal Functionality

The *CPP* describes the specific capabilities that a *Trading Partner* supports as well as the *Service Interface* requirements that need to be met in order to exchange business documents with that *Trading Partner*. The *CPP* contains essential information about the *Trading Partner* including, but not limited to: contact information, industry classification, supported *Business Processes*, *Interface* requirements and *Messaging Service* requirements. *CPP's* MAY also contain security and other implementation specific details. Each ebXML compliant *Trading Partner* SHOULD register their *CPP(s)* in an ebXML compliant *Registry Service*, thus providing a discovery mechanism that allows *Trading Partners* to (1) find one another, (2) discover the *Business Process* that other *Trading Partners* support.

The *CPP* definition SHALL provide for unambiguous selection of choices in all instances where there may be multiple selections (e.g. HTTP or SMTP transport).

9.1.3 CPA Formal Functionality

A *Collaboration Protocol Agreement* (*CPA*) is a document that represents the intersection of two *CPP's* and is mutually agreed upon by both Trading Partners who wish to conduct *e-Business* using ebXML.

A *CPA* describes: (1) the *Messaging Service* and (2) the *Business Process* requirements that are agreed upon by two or more *Trading Partners*. Conceptually, ebXML supports a three level view of narrowing subsets to arrive at *CPA's* for transacting *e-Business*. The outer-most scope relates to all of the capabilities that a *Trading Partner* can support, with a subset of what a *Trading Partner* "will" actually support.

A *CPA* contains the *Messaging Service Interface* requirements as well as the implementation details pertaining to the mutually agreed upon *Business Processes* that both *Trading Partners* agree to use to conduct *e-Business*. *Trading Partners* may decide to register their *CPA's* in an ebXML compliant *Registry Service*, but this is not a mandatory part of the *CPA* creation process.

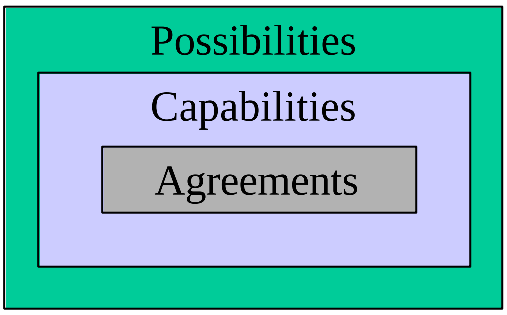


Figure 6 Three level view of CPA's

Business Collaborations are the first order of support that can be claimed by ebXML *Trading Partners*. This "claiming of support" for specific *Business Collaborations* is facilitated by a distinct profile defined specifically for publishing or advertising in a directory service, such as an ebXML *Registry* or other available service. Figure 3 below outlines the scope for *Collaboration Protocol Agreements* within ebXML.

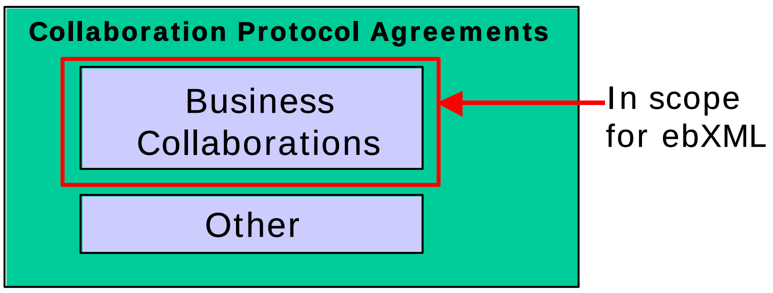


Figure 7 Scope for CPA's

The *CPA-CPP* specification includes a non-normative appendix that discusses *CPA* composition and negotiation and includes advice as to composition and negotiation procedures.

9.1.4 CPP Interfaces

Interface to Business Processes

A *CPP* SHALL be capable of referencing one or more *Business Processes* supported by the *Trading Partner* owning the *CPP* instance. The *CPP* SHALL reference the Roles within a *Business Process* that the user is capable of assuming. An example of a Role could be the notion of a "Seller" and "Buyer" within a "Purchasing" *Business Process*.

The *CPP* SHALL be capable of being stored and retrieved from an ebXML *Registry* Mechanism

A *CPP* SHOULD also describe binding details that are used to build an ebXML *Message Header*.

9.1.5 CPA Interfaces

A *CPA* governs the *Business Service Interface* used by a *Trading* Partner to constrain the *Business Service Interface* to a set of parameters agreed to by all *Trading Partners* who will execute such an agreement.

CPA's have *Interfaces* to *CPP's* in that the *CPA* is derived through a process of mutual negotiation narrowing the *Trading Partners* capabilities (*CPP*) into what the *Trading Partner* "will" do (*CPA*).

A *CPA* must reference to a specific *Business Process* and the interaction requirements needed to execute that *Business Process*.

A *CPA* MAY be stored in a *Registry* mechanism, hence an implied ability to be stored and retrieved is present.

9.1.6 Non-Normative Implementation Details

A *CPA* is negotiated after the Discovery and Retrieval Phase and is essentially a snapshot of the *Messaging Services* and *Business Process* related information that two or more *Trading Partners* agree to use to exchange business information. If any parameters contained within an accepted *CPA* change after the agreement has been executed, a new *CPA* SHOULD be negotiated between *Trading Partners*.

In some circumstances there may be a need or desire to describe casual, informal or implied *CPA's*.

An eventual goal of ebXML is to facilitate fully automated *CPA* generation. In order to meet this goal, a formal methodology SHOULD be specified for the *CPA* negotiation process.

9.2 Business Process and Information Modelling

9.2.1 Introduction

The ebXML *Business Process and Information Meta Model* is a mechanism that allows *Trading Partners* to capture the details for a specific business scenario using a consistent modelling methodology. A *Business Process* describes in detail how *Trading Partners* take on roles, relationships and responsibilities to facilitate interaction with other *Trading Partners* in shared collaborations. The interaction between roles takes place as a choreographed set of business transactions. Each business transaction is expressed as an exchange of electronic *Business Documents*. *Business Documents* MAY be composed from re-useable *Business Information Objects* (see "Relationships to Core Components" under 8.2.3 "*Interfaces*" below). At a lower level, *Business Processes* can be composed of re-useable *Core Processes*, and *Business Information Objects* can be composed of re-useable *Core Components*.

The ebXML *Business Process and Information Meta Model* supports requirements, analysis and design viewpoints that provide a set of semantics (vocabulary) for each viewpoint and forms the basis of specification of the artefacts that are required to facilitate *Business Process* and information integration and interoperability.

An additional view of the *Meta Model*, the *Specification Schema*, is also provided to support the direct specification of the set of elements required to configure a runtime system in order to execute a set of ebXML business transactions. By drawing out modelling elements from several of the other views, the *Specification Schema* forms a semantic subset of the ebXML *Business Process and Information Meta Model.* The *Specification Schema* is available in two stand-alone representations, a *UML* profile, and a DTD.

The relationship between the ebXML *Business Process and Information Meta Model* and the ebXML *Specification Schema* can be shown as follows:

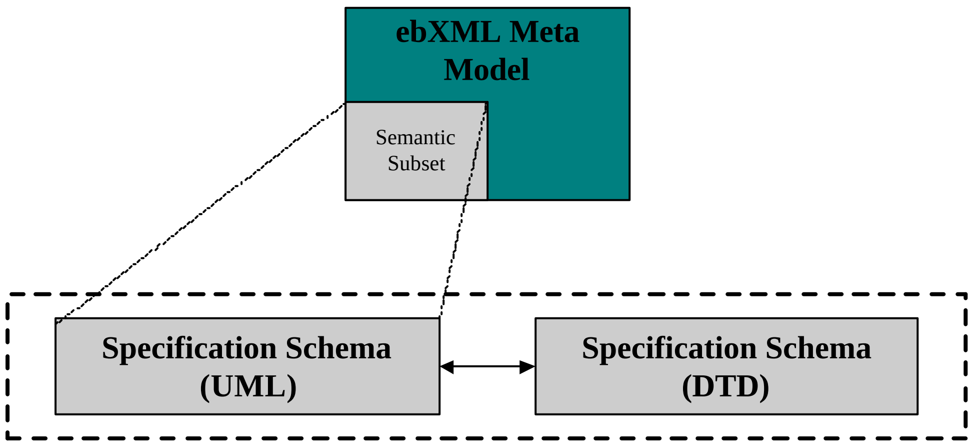


Figure 8 ebXML Meta Model - Semantic Subset

The *Specification Schema* supports the specification of business transactions and the choreography of business transactions into *Business Collaborations*. Each *Business Transaction* can be implemented using one of many available standard patterns. These patterns determine the actual exchange of *Messages* and signals between *Trading Partners* to achieve the required electronic transaction. To help specify the patterns *the Specification Schema* is accompanied by a set of standard patterns, and a set of modelling elements common to those patterns. The full specification of a *Business Process* consists of a *Business Process and Information Meta Model* specified against the *Specification Schema* and an identification of the desired pattern(s). This information serves as the primary input for the formation of *Collaboration Protocol Profiles* (*CPP's*) and *CPA's*. This can be shown as follows:

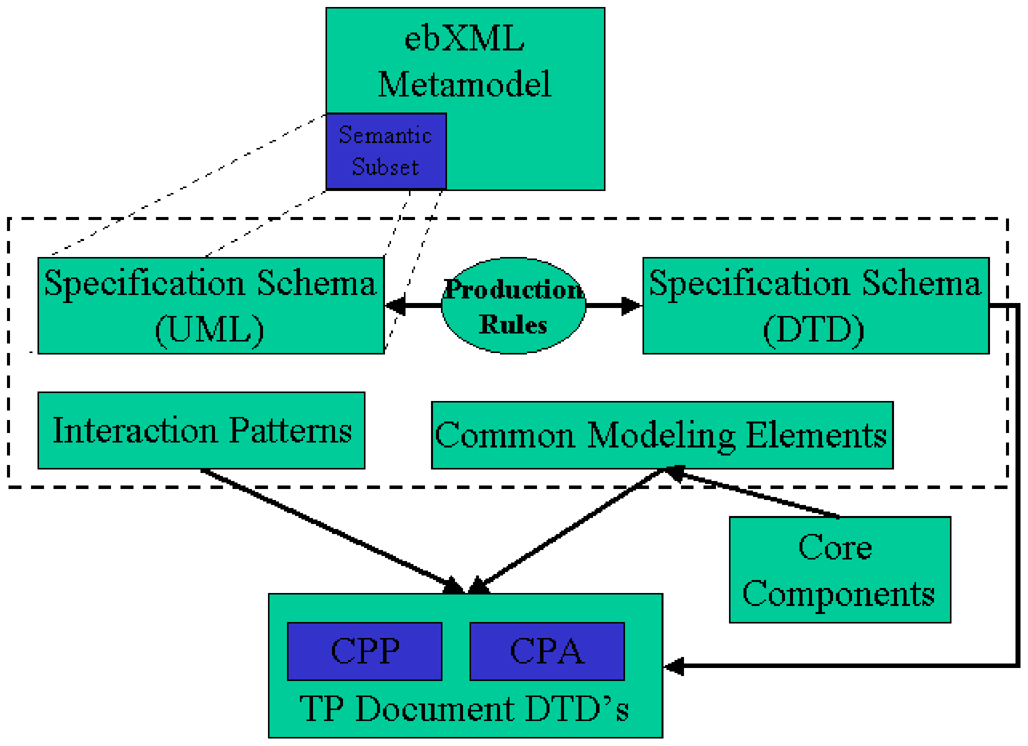


Figure 9 ebXML Meta Model

There are no formal requirements to mandate the use of a modelling language to compose new *Business Processes*, however, if a modelling language is used to develop *Business Processes*, it SHALL be the *Unified Modelling Language (UML)*. This mandate ensures that a single, consistent modelling methodology is used to create new *Business Processes*. One of the key benefits of using a single consistent modelling methodology is that it is possible to compare models to avoid duplication of existing *Business Processes*.

To further facilitate the creation of consistent *Business Processes* and information models, ebXML will define a common set of *Business Processes* in parallel with a *Core Library*. It is possible that users of the ebXML infrastructure may wish to extend this set or use their own *Business Processes*.

9.2.2 Formal Functionality

The representation of a *Business Process* document instance SHALL be in a form that will allow both humans and applications to read the information. This is necessary to facilitate a gradual transition to full automation of business interactions.

The *Business Process* SHALL be storable and retrievable in a *Registry* mechanism. *Business Processes* MAY be registered in an ebXML *Registry* in order to facilitate discovery and retrieval.

To be understood by an application, a *Business Process* SHALL be expressible in *XML* syntax. A *Business Process* MAY be constructed as an *Business Process and Information Meta Model* or an *XML* representation of that model. *Business Processes* are capable of expressing the following types of information:

- Choreography for the exchange of document instances. (e.g. the choreography of necessary *Message* exchanges between two Trading Partners executing a “Purchasing” ebXML transaction.)
- References to *Business Process and Information Meta Model* or *Business Documents* (possibly *DTD’s* or *Schemas*) that add structure to business data.
- Definition of the roles for each participant in a *Business Process.*

A Business Process:

- Provides the contextual constraints for using *Core Components*
- Provides the framework for establishing *CPAs*
- Specifies the domain owner of a *Business Process*, along with relevant contact information.
  [NOTE: the above lists are not inclusive.]

9.2.3 Interfaces

Relationship to CPP and CPA

The *CPP* instance of a *Trading Partner* defines that partner’s functional and technical capability to support zero, one, or more *Business Processes* and one or more roles in each process.

The agreement between two *Trading Partners* defines the actual conditions under which the two partners will conduct business transactions together. The *Interface* between the *Business Process,* its *Information Meta Model,* and the *CPA* is the part of the *Business Process* document. This MAY be instantiated as an *XML* document representing the business transactional and collaboration layers of the *Business Process and Information Meta Model*. The expression of the sequence of commercial transactions in *XML* is shared between the *Business Process* and *Trading Partner Information* models.

Relationship to Core Components

A *Business Process* instance SHOULD specify the constraints for exchanging business data with other *Trading Partners*. The business information MAY be comprised of components of the ebXML *Core Library*. A *Business Process* document SHALL reference the *Core Components* directly or indirectly using a *XML* document that references the appropriate Business and Information Models and/or Business Documents (possibly DTD's or Schemas). The mechanism for interfacing with the *Core Components* and *Core Library* SHALL be by way of a unique identifier for each component.

Relationship to ebXML Messaging

A *Business Process* instance SHALL be capable of being transported from a *Registry Service* to another *Registry Service* via an ebXML *Message*. It SHALL also be capable of being transported between a *Registry* and a users application via the ebXML *Messaging Service*.

Relationship to a Registry System

A *Business Process* instance intended for use within the ebXML infrastructure SHALL be retrievable through a *Registry* query, and therefore, each *Business Process* SHALL contain a unique identifier.

9.2.4 Non-Normative Implementation Details

The exact composition of *Business Information Objects* or a *Business Document* is guided by a set of contexts derived from the *Business Process*. The modelling layer of the architecture is highlighted in green in Figure 6 below.

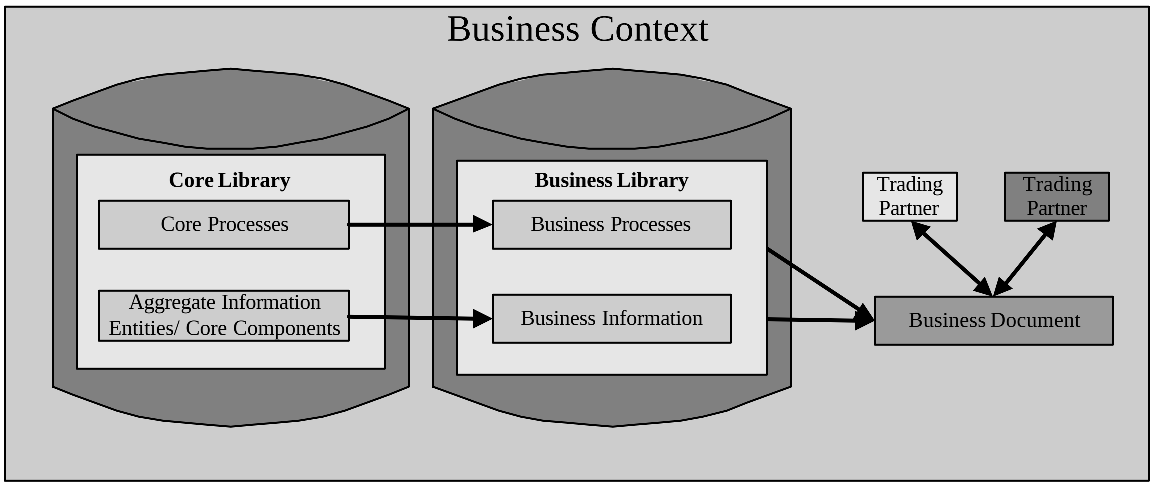


Figure 10 ebXML Business Process and Information Modelling layer

ebXML *Business Process and Information Meta Model* MAY be created following the recommended UN/CEFACT *Modelling Methodology* (*UMM*), or MAY be arrived at in any other way, as long as they comply with the ebXML *Business Process and Information Meta Model*.

9.3 Core Components and Core Library Functionality

9.3.1 Introduction

A *Core Component* captures information about a real world business concept, and the relationships between that concept, other *Business Information Objects,* and a contextual description that describes how a *Core* or *Aggregate Information Entity* may be used in a particular ebXML *e-Business* scenario.

A *Core Component* can be either an individual piece of business information, or a natural "go-together" family of *Business Information Objects* that may be assembled into *Aggregate Information Entities*.

The ebXML Core Components project team SHALL define an initial set of *Core Components*. ebXML users may adopt and/or extend components from the ebXML *Core Library.*

9.3.2 Formal Functionality

As a minimum set of requirements, *Core Components* SHALL facilitate the following functionality:

Core Components SHALL be storable and retrievable using an ebXML *Registry* Mechanism.

Core Components SHALL capture and hold a minimal set of information to satisfy *e-Business* needs.

Core Components SHALL be capable of being expressed in *XML* syntax.

A *Core Component* SHALL be capable of containing:

- Another *Core Component* in combination with one or more individual pieces of *Business Information Objects*.
- Other *Core Components* in combination with zero or more individual pieces of *Business Information Objects*.

A *Core Component* SHALL be able to be uniquely identified.

9.3.3 Interfaces

A *Core Component* MAY be referenced indirectly or directly from a *Business Document* instance. The *Business Process* MAY specify a single or group of *Core Components* as required or optional information as part of a *Business Document* instance.

A *Core Component* SHALL interface with a *Registry* mechanism by way of being storable and retrievable in such a mechanism.

A *Core Component* MAY interface with an *XML* Element from another *XML* vocabulary by the fact it is bilaterally or unilaterally referenced as a semantic equivalent.

9.3.4 Non-Normative Implementation Details

A *Core Component* MAY contain attribute(s) or be part of another *Core Component*, thus specifying the precise context or combination of contexts in which it is used.

The process of aggregating *Core Components* for a specific business context, shall include a means to identify the placement of a *Core Component* within another *Core Component*. It MAY also be a combination of structural contexts to facilitate *Core Component* re-use at different layers within another *Core Component* or *Aggregate Information Entity.* This is referred to as *Business Context*.

Context MAY also be defined using the *Business Process and Information Meta Model*, which defines the instances of *Business Information Objects* in which the *Core Component* occurs.

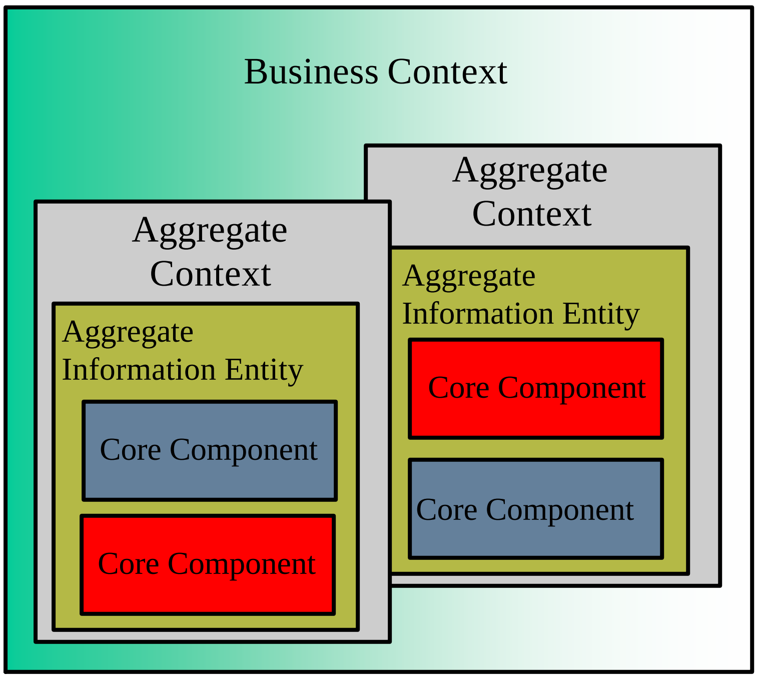


Figure 11 Business Context defined in terms of Aggregate Context, Aggregate Information Entities, and Core Components

The pieces of *Business Information Objects*, or *Core Components*, within a generic *Core Component* may be either mandatory, or optional. A *Core Component* in a specific context or combination of contexts (aggregate or business context) may alter the fundamental mandatory/optional cardinality.

9.4 Registry Functionality

9.4.1 Introduction

An ebXML *Registry* provides a set of services that enable the sharing of information between *Trading Partners*. A *Registry* is a component that maintains an interface to metadata for a registered item. Access to an ebXML *Registry* is provided through *Interfaces* (APIs) exposed by *Registry Services*.

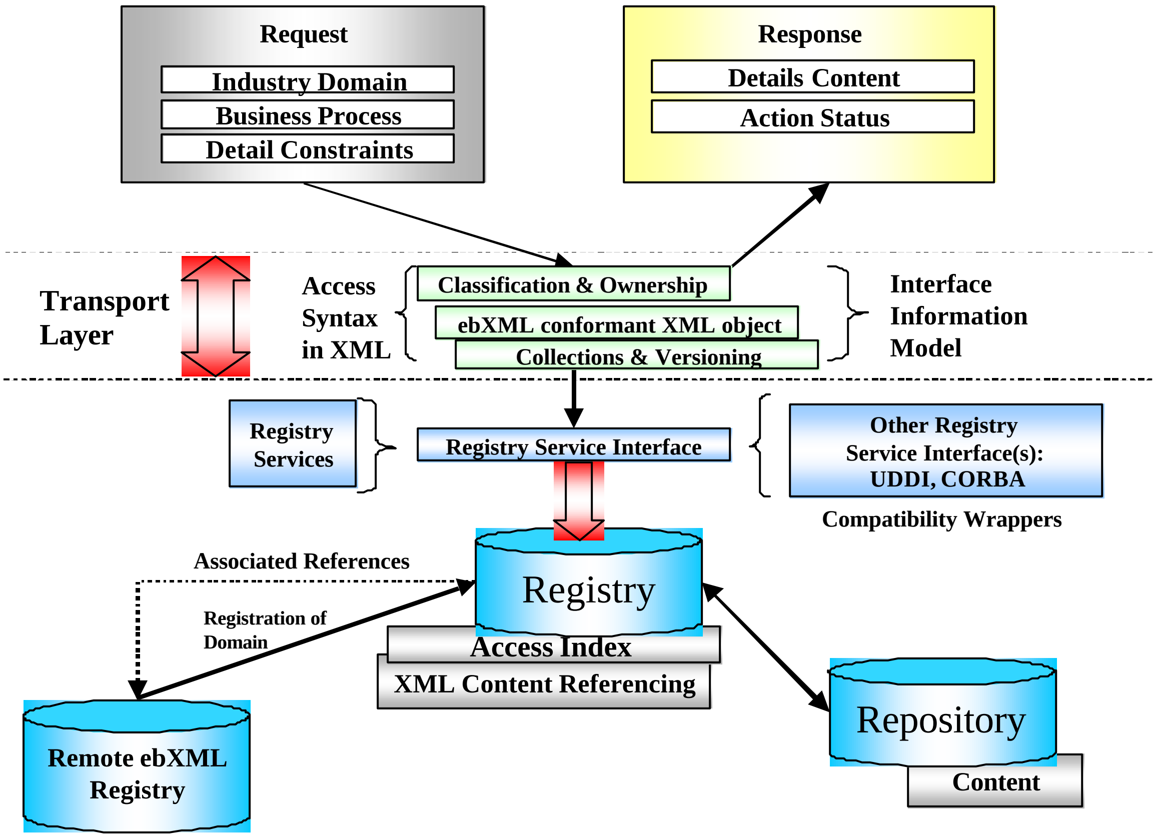


Figure 12 Overall Registry Architecture.

9.4.2 Formal Functionality

A *Registry* SHALL accommodate the storage of items expressed in syntax using multi-byte character sets.

Each *Registry Item*, at each level of granularity as defined by the *Submitting Organization*, MUST be uniquely identifiable. This is essential to facilitate application-to-Registry queries.

A *Registry* SHALL return either zero or one positive matches in response to a contextual query for a unique identifier. In such cases where two or more positive results are displayed for such queries, an error message SHOULD be reported to the *Registry Authority*.

A *Registry Item* SHALL be structured to allow information associations that identify, name, describe it, give its administrative and access status, define its persistence and mutability, classify it according to pre-defined classification schemes, declare its file representation type, and identify the submitting and responsible organizations.

The *Registry Interface* serves as an application-to-registry access mechanism. Human-to-registry interactions SHALL be built as a layer over a *Registry Interface* (e.g. a Web browser) and not as a separate *Interface*.

The *Registry Interface* SHALL be designed to be independent of the underlying network protocol stack (e.g. HTTP/SMTP over TCP/IP). Specific instructions on how to interact with the *Registry Interface* MAY be contained in the payload of the ebXML *Message*.

The processes supported by the *Registry* MAY also include:

- A special *CPA* between the *Registry* and *Registry Clients*.

- A set of functional processes involving the *Registry* and *Registry Clients*.
- A set of *Business Messages* exchanged between a *Registry Client* and the *Registry* as part of a specific *Business Process*.
- A set of primitive *Interface* mechanisms to support the *Business Messages* and associated query and response mechanisms.
- A special *CPA* for orchestrating the interaction between ebXML compliant Registries.
- A set of functional processes for *Registry*-to-*Registry* interactions.
- A set of error responses and conditions with remedial actions.

To facilitate the discovery process, browse and drill down queries MAY be used for human interactions with a *Registry* (e.g. via a Web browser). A user SHOULD be able to browse and traverse the content based on the available *Registry* classification schemes.

Registry Services exist to create, modify, and delete *Registry Items* and their metadata.

Appropriate security protocols MAY be deployed to offer authentication and protection for the *Repository* when accessed by the *Registry*.

Unique Identifiers (*UIDs*) SHALL be assigned to all items within an ebXML *Registry System*. *UID* keys are REQUIRED references for all ebXML content. *Universally Unique Identifiers (UUIDs)* MAY be used to ensure that *Registry* entries are truly globally unique, and thus when systems query a *Registry* for a *UUID*, one and only one result SHALL be retrieved.

To facilitate semantic recognition of *Business Process and Information Meta Models*, the *Registry Service* SHALL provide a mechanism for incorporating human readable descriptions of *Registry* items. Existing *Business Process and Information Meta Models* (e.g. RosettaNet PIPs) and *Core Components* SHALL be assigned *UID* keys when they are registered in an ebXML compliant *Registry Service*. These *UID* keys MAY be implemented in physical *XML* syntax in a variety of ways. These mechanisms MAY include, but are not limited to:

- A pure explicit reference mechanism (example: URN:*UID* method),
- A referential method (example: URI:*UID* / namespace:*UID*),
- An object-based reference compatible with W3C Schema (*example* URN:complextype name), and
- A data type based reference (example: ISO 8601:2000 Date/Time/Number data typing and then legacy data typing).

Components in ebXML MUST facilitate multilingual support. A *UID* reference is particularly important here as it provides a language neutral reference mechanism. To enable multilingual support, the ebXML specification SHALL be compliant with Unicode and ISO/IEC 10646 for character set and UTF-8 or UTF-16 for character encoding.

9.4.3 Interfaces

ebXML Messaging:

The query syntax used by the *Registry* access mechanisms is independent of the physical implementation of the backend system.

The ebXML *Messaging Service* MAY serve as the transport mechanism for all communication into and out of the *Registry*.

Business Process:

Business Processes are published and retrieved via ebXML *Registry Services*.

Core Components:

Core Components are published and retrieved via ebXML *Registry Services*.

Any item with metadata: *XML* elements provide standard metadata about the item being managed through ebXML *Registry Services*. Since ebXML Registries are distributed each *Registry* MAY interact with and cross-reference another ebXML *Registry*.

9.4.4 Non-Normative Implementation Details

The *Business Process and Information Meta Models* within a *Registry* MAY be stored according to various classification schemes.

The existing ISO11179/3 work on *Registry* implementations MAY be used to provide a model for the ebXML *Registry* implementation.

Registry Items and their metadata MAY also be addressable as *XML* based URI references using only HTTP for direct access.

Examples of extended *Registry Services* functionality may be deferred to a subsequent phase of the ebXML initiative. This includes, but is not limited to transformation services, workflow services, quality assurance services and extended security mechanisms.

A *Registry Service* MAY have multiple deployment models as long as the *Registry Interfaces* are ebXML compliant.

The *Business Process and Information Meta Model* for an ebXML *Registry Service* may be an extension of the existing OASIS Registry/Repository Technical Specification, specifically tailored for the storage and retrieval of business information, whereas the OASIS model is a superset designed for handling extended and generic information content.

9.5 Messaging Service Functionality

9.5.1 Introduction

The ebXML *Message Service* mechanism provides a standard way to exchange business *Messages* among ebXML *Trading Partners*. The ebXML *Messaging Service* provides a reliable means to exchange business *Messages* without relying on proprietary technologies and solutions. An ebXML *Message* contains structures for a *Message Header* (necessary for routing and delivery) and a *Payload* section.

The ebXML *Messaging Service* is conceptually broken down into three parts: (1) an abstract *Service Interface*, (2) functions provided by the *Messaging Service Layer*, and (3) the mapping to underlying transport service(s). The relation of the abstract *Interface*, *Messaging Service Layer*, and transport service(s) are shown in Figure 13 below.

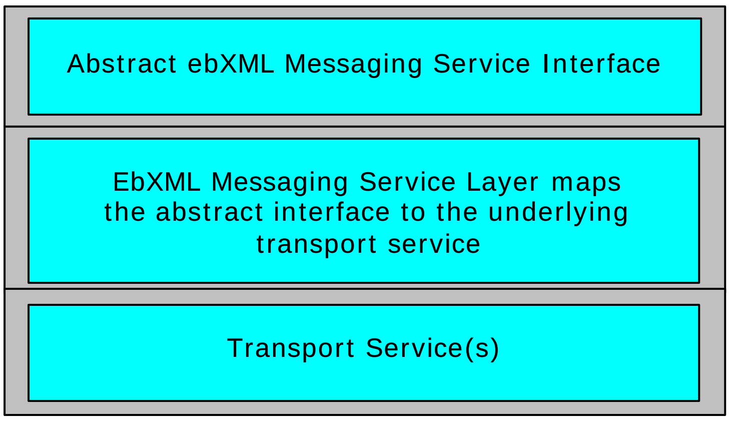

Abstract ebXML Messaging Service Interface
EbXML Messaging Service Layer maps the abstract interface to the underlying transport service
Transport Service(s)

Figure 13 ebXML Messaging Service

The following diagram depicts a logical arrangement of the functional modules that exist within the ebXML *Messaging Services* architecture. These modules are arranged in a manner to indicate their inter-relationships and dependencies. This architecture diagram illustrates the flexibility of the ebXML *Messaging Service*, reflecting the broad spectrum of services and functionality that may be implemented in an ebXML system.

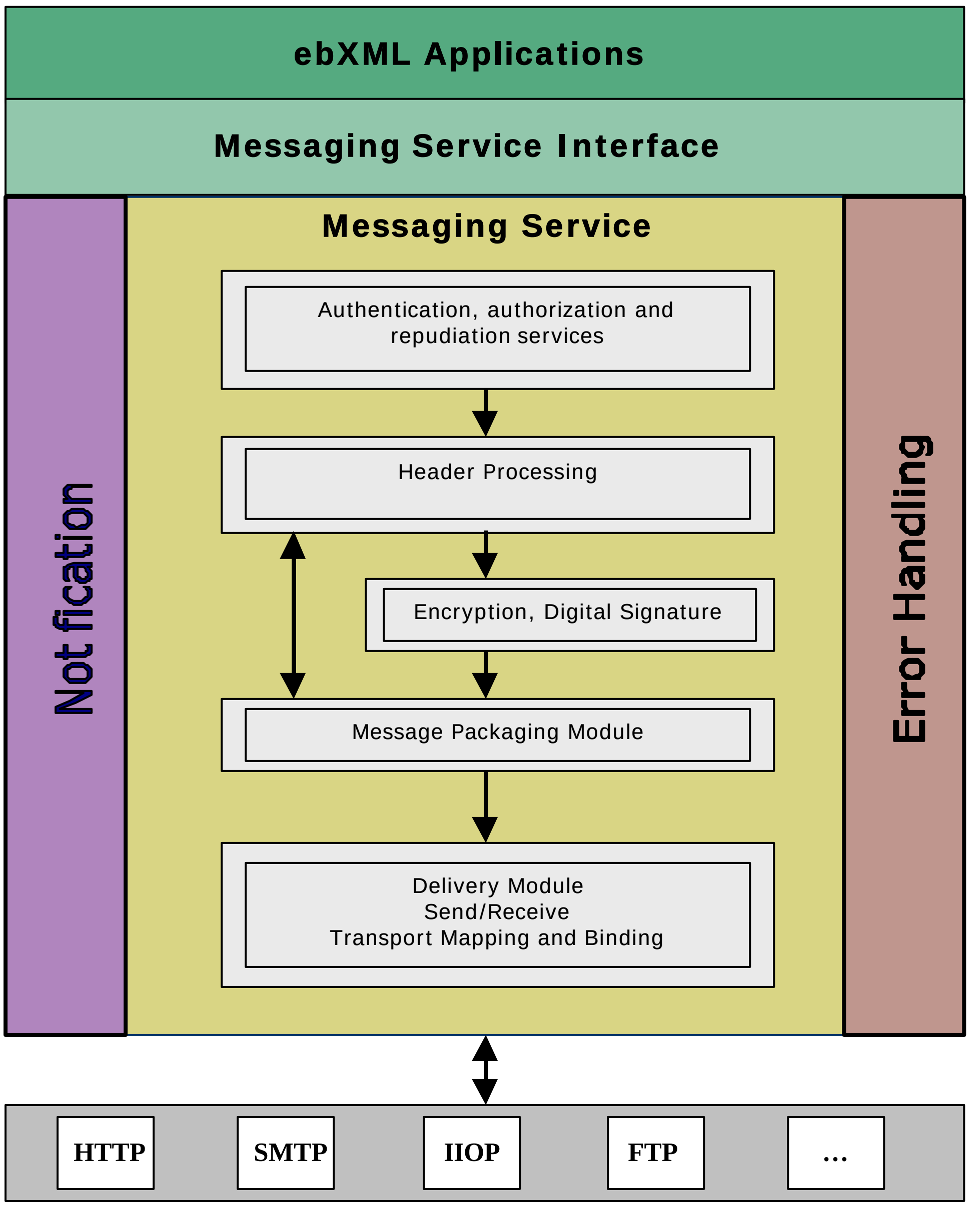


Figure 14 The Messaging Service Architecture

9.5.2 Formal Functionality

The ebXML *Messaging Service* provides a secure, consistent and reliable mechanism to exchange ebXML *Messages* between users of the ebXML infrastructure over various transport *Protocols* (possible examples include SMTP, HTTP/S, FTP, etc.).

The ebXML *Messaging Service* prescribes formats for all *Messages* between distributed ebXML *Components* including *Registry* mechanisms and compliant user *Applications*.

The ebXML *Messaging Service* does not place any restrictions on the content of the payload.

The ebXML *Messaging Service* supports simplex (one-way) and request/response (either synchronous or asynchronous) *Message* exchanges.

The ebXML *Messaging Service* supports sequencing of payloads in instances where multiple payloads or multiple *Messages* are exchanged between *Trading Partners*.

The ebXML *Messaging Service Layer* enforces the "rules of engagement" as defined by two *Trading Partners* in a *Collaboration Protocol Agreement* (including, but not limited to security and *Business Process* functions related to *Message* delivery). The *Collaboration Protocol Agreement* defines the acceptable behaviour by which each *Trading Partner* agrees to abide. The definition of these ground rules can take many forms including formal *Collaboration Protocol Agreements*, interactive agreements established at the time a business transaction occurs (e.g. buying a book online), or other forms of agreement. There are *Messaging Service Layer* functions that enforce these ground rules. Any violation of the ground rules result in an error condition, which is reported using the appropriate means.

The ebXML *Messaging Service* performs all security related functions including:

- Identification
- Authentication (verification of identity)
- Authorization (access controls)
- Privacy (encryption)
- Integrity (message signing)
- Non-repudiation
- Logging

9.5.3 Interfaces

The ebXML *Messaging Service* provides ebXML with an abstract *Interface* whose functions, at an abstract level, include:

- Send – send an ebXML *Message* – values for the parameters are derived from the ebXML *Message Headers*.
- Receive – indicates willingness to receive an ebXML *Message*.
- Notify – provides notification of expected and unexpected events.
- Inquire – provides a method of querying the status of the particular ebXML *Message* interchange.

The ebXML *Messaging Service* SHALL interface with internal systems including:

- Routing of received *Messages* to internal systems
- Error notification

The ebXML *Messaging Service* SHALL help facilitate the *Interface* to an ebXML *Registry*.

9.5.4 Non-Normative Implementation Details

ebXML Message Structure and Packaging

Figure below illustrates the logical structure of an ebXML *Message*.

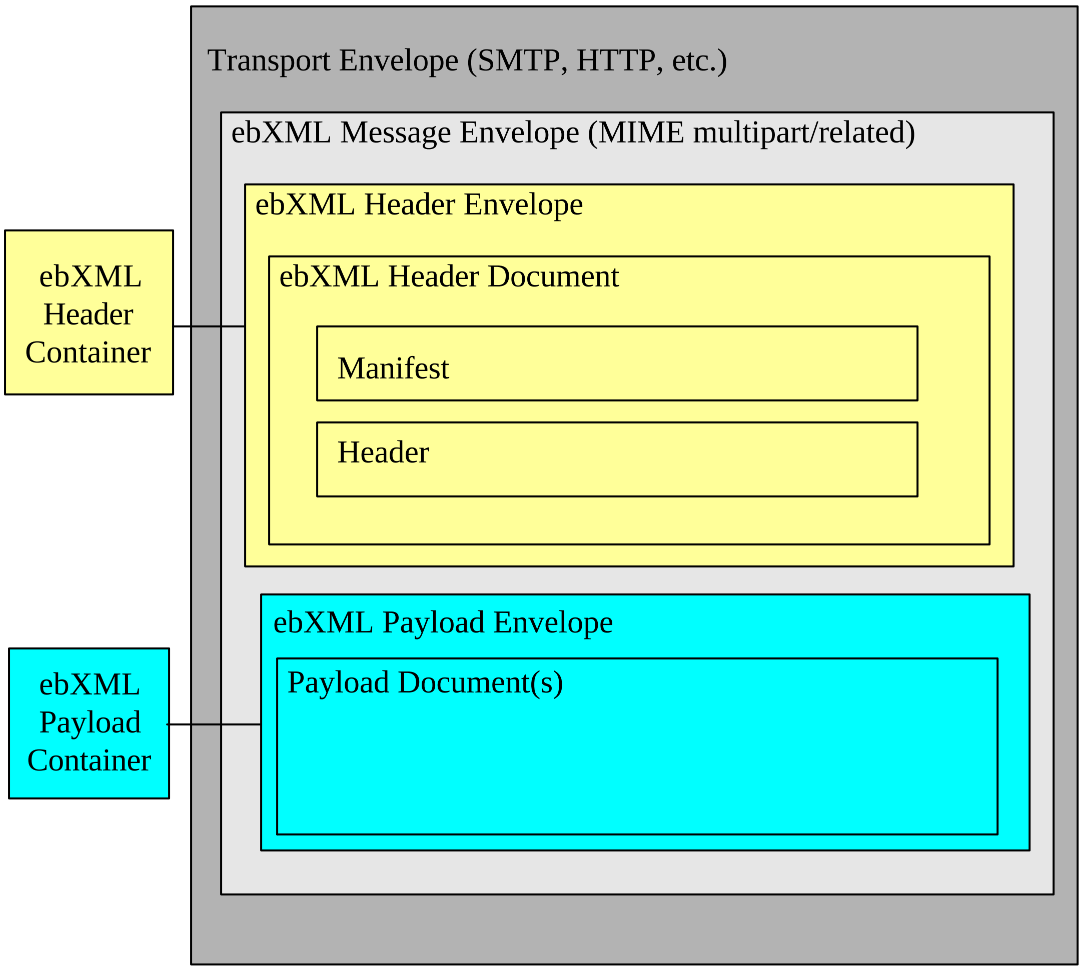


Figure 15 ebXML Message Structure

An ebXML Message consists of an optional transport Protocol specific outer Communication Protocol Envelope and a Protocol independent ebXML Message Envelope. The ebXML Message Envelope is packaged using the MIME multipart/related content type. MIME is used as a packaging solution because of the diverse nature of information exchanged between Partners in e-Business environments. For example, a complex Business Transaction between two or more Trading Partners might require a payload that contains an array of business documents (XML or other document formats), binary images, or other related Business Information.

10 Wireless technologies

REGNET can offer a WAP based access to the REGNET System, providing a novel access method for: tele-shopping, e-Business, database browsing, and virtual visit to museum or other masterpieces. REGNET will examine how next generation mobile networks can be exploited to widen the potentiality of WEB services in the field of Cultural Heritage. As an example, services can be used by interacting with the display of the mobile device (download of images or business information) or, in future extensions of REGNET Interface, through voice (content description, service centre). For the development of WAP services, WML (Wireless Markup Language) is the language enabling web browsing, whereas VoiceXML addresses voice applications. Appropriate gateway functions and interworking units will be envisaged to appropriately interface the REGNET architecture with the GPRS (General Packet Radio Service) and UMTS (Universal Mobile Telecommunication Systems) structures and ensure a seamless provisioning of REGNET services to mobile customers. Furthermore, the possibility of having access to the “mobile community” will enlarge the set of services that will be part of the REGNET demonstrator.

This paragraph aims at describing the technologies related to the development of the wireless environment in REGNET platform. It covers the WAP architecture, the development issues in the WAP Application Environment (WAE) and the main characteristics of the bearers taken into account.

Sections 10.4.1 and 10.4.3 describe the overall features of GPRS and UMTS respectively, whilst sections 10.4.2 and 10.4.4 present some concepts about their use as WAP bearer. The Bluetooth standard was also take into consideration in section 10.5 as an enabling technology for future REGNET services, as it introduces some interesting scenarios for short-range wireless communications. It is a good candidate to enter the Technological Implementation Plan. Some notes on its use as WAP bearer are reported in section 10.5.1.

10.1 Wireless Application Protocol (WAP)

The Wireless Application Protocol (WAP) is a hot topic that has been widely hyped in the mobile industry and outside of it. WAP is simply a standardized way used by a mobile phone to talk to a server installed in the mobile phone network.

WAP embraces and extends the previously conceived and developed wireless data protocols. Phone.com created a version of the standard HTML (HyperText Markup Language) Internet protocols designed specifically for effective and cost-effective information transfer across mobile networks. Wireless terminals incorporated a HDML (Handheld Device Markup Language) microbrowser, and Phone.com's Handheld Device Transport Protocol (HDTP) then linked the terminal to the UP.Link Server Suite that connected to the Internet or intranet where the information being requested resides. The Internet site content was tagged with HDML.

WAP is a hot topic for several reasons:

- It provides a standardized way of linking the Internet to mobile phones, thereby linking two of the hottest industries anywhere.
- Its founder members include the major wireless vendors of Motorola, Nokia, and Ericsson.
- By April 2000, the WAP Forum had over 350 member companies.

Mobile information services have not been as successful as many network operators expected. WAP is seen as a way to rectify this situation.

WAP also has its detractors and controversies:

- Compared with the installed base of Short Message Service (SMS) compliant phones, the relative number of handsets supporting WAP is tiny. WAP is a protocol that runs on top of an underlying bearer. None of the existing GSM bearers for WAP- the Short Message Service (SMS), Unstructured Supplementary Services Data (USSD) and Circuit Switched Data (CSD) are optimised for WAP.
- There are many WAP Gateway vendors out there competing against each other with largely the same standardized product.
- Other protocols such as SIM Application Toolkit and Mobile Station Application Execution Environment (MExE) are respectively already widely supported or designed to super cede WAP.
- WAP services are expected to be expensive to use since the tendency is to be on-line for a long Circuit Switched Data (CSD) call as the end user uses features such as

interactivity and selection of more information. Without specific tariff initiatives, there are likely to be some surprised WAP users when they see their mobile phone bill for the first time after starting using WAP.

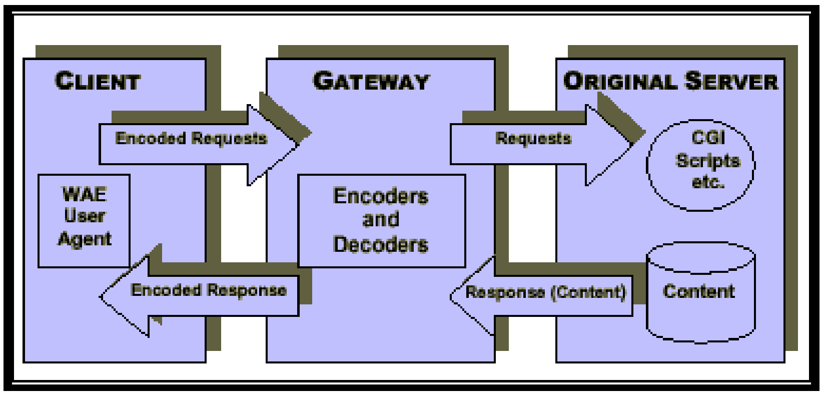


Figure 16 WAP Architecture

The WAP Gateway is essentially a piece of middleware, taking information from a web server, processing it, and sending it out over the mobile network to a WAP client.

Someone with a WAP-compliant phone uses the in-built microbrowser to:

1. Make a request in WML (Wireless Markup Language), a language derived from HTML especially for wireless network characteristics.
2. This request is passed to a WAP Gateway that either then retrieves the information from an Internet server in standard HTML format or preferably directly prepared for wireless terminals using WML. If the content being retrieved is in HTML format, a filter in the WAP Gateway may try to translate it into WML. A WML scripting language is available to format data such as calendar entries and electronic business cards for direct incorporation into the client device.
3. The requested information is then sent from the WAP Gateway to the WAP client, using whatever mobile network bearer service is available and most appropriate.

There are several vendors of WAP Gateways that network operators, content providers and application developers can work with to develop WAP-based services. WAP Gateways are installed into the mobile phone network to provide a gateway between the Internet and different mobile non-voice services such as the Short Message Service, Circuit Switched Data and General Packet Radio Service.

10.2 Voice Services

Making use of the VoiceXML language is possible to develop services that allow making a call from any phone (wire-line or wireless) and retrieves information from the Internet or performs transactions.

Some typologies of voice services are the following:

- Web portal-type information (weather, stocks, news, sports, etc.);
- e-Business
- Intranet applications (e.g. calendar, tasks).

In VoiceXML a voice service can be described as a finite state machine or in other words as a sequence of interaction dialogs between a user and an implementation platform.

The user is always in one conversational state or dialogs at a time and every time a user makes an actions (or automatically) there is a transition to another state. Transitions are specified using URIs which define the next document and dialog to use. Execution is terminated when a dialog does not specify a successor, or if it has an element that explicitly exits the conversation.

Architectural model

At a high level, the architectural model for Voice services compared to that for Web services is the following:

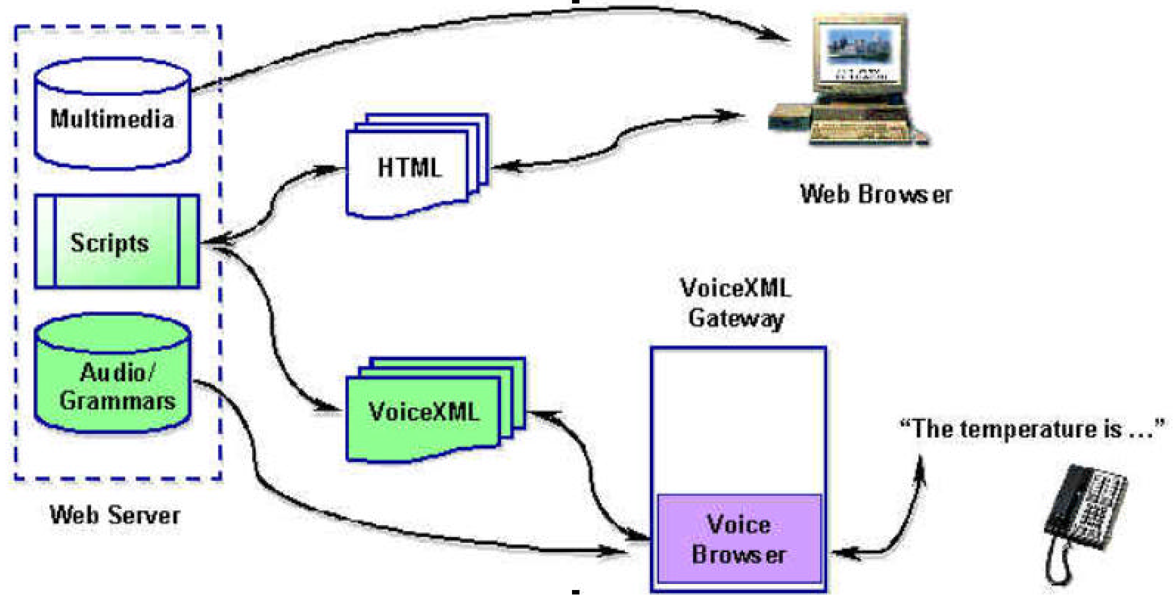


Figure 17 Voice service Architectural Model

Voice services reside on the Web server. The access to voice applications over the phone is done through the VoiceXML Gateway that function as the liaison between a telephone line and Internet. The VoiceXML Gateway is a computer server connected to the Public Switched Telephone Network (PSTN) via phone lines and to the Internet via an Internet connection.

The VoiceXML Gateway includes Voice Browser software, Automatic Speech Recognition (ASR) software, and Text-to-Speech (TTS) software.

The Voice Browser, closely analogous to a graphical web browser (e.g., Netscape, Internet Explorer) interacts with the user via dialogue specifications fetched from the Internet (VXML pages) rather than via HTML pages.

The output is rendered via recorded prompts (fetched from Web or in local files (.wav)); or via text by text-to-speech (TTS) engine. Instead, input are fetched via DTMF key presses or via spoken words recognized by an automatic speech recognition (ASR) engine (matched with grammars internal or external).

10.3 Development standards

10.3.1 WML

WML (Wireless Markup Language) is a markup language based on XML. The WML official specification is developed and maintained by the WAP Forum (an industry-wide consortium founded by Nokia, Motorola, and Ericsson) and defines the syntax, variables, and elements used in a valid WML file.

WML is the language used to create applications that run on cell phones and other wireless devices. It was designed for low-bandwidth, small-display devices including cellular phones and pagers.

If a phone or other communications device is said to be WAP-capable, means that it has a piece of software loaded onto it (known as a microbrowser) that fully understands how to handle all entities in the WML DTD.

A single WML document (i.e. the elements contained within the <wml> document element) is known as a deck. A single interaction between a user agent and a user is known as a card. Depending on your client's memory capabilities, it may be necessary to split multiple cards up into multiple decks to prevent a single deck from becoming too large.

Hello World

<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN
http://www.wapforum.org/DTD/wml_1.1.xml">
 	
      <wml>
        <card id="Card1" title="Wap-UK.com">
          <p>
            <!-- Hello World example -->
            Hello World
          </p>
        </card>
      </wml>

Table 1 Example of a very simple WML file

10.3.2 VoiceXML

VoiceXML, is a Web-base markup language for voice applications based on eXtensible Markup Language (XML). It is a specification of the VoiceXML Forum, an industry organization founded by AT&T, IBM, Lucent and Motorola, and chartered with establishing and promoting the VoiceXML.

It's major goals are:

- bring the advantages of web-base development and content delivery to interactive voice response applications;
- enables integration of voice services with data services using the familiar client-server paradigm.

While HTML assumes a graphical web browser, with display, keyboard, and mouse, VoiceXML assumes a voice browser with audio output (computer-synthesized and/or recorded), and audio input (voice and/or keypad tones).

Example of a VoiceXML page:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<vxml version="1.0">
<form id="ask_the_city">
<field name="city">
  <audio src="ask_the_city.wav"/>
  <grammar src="cities.grammar" />
  <filled>
    <goto next="ask_the_day.jsp?theCity=city"/>
  </filled>
</field>
</form>
</vxml>
```

10.3.3 Java 2 Micro Edition

In the post-PC era, the market for network-connected devices continues to grow at an enormous rate. While new classes of devices like smart cellular telephones, pagers, and PDAs proliferate, traditional consumer electronics including televisions, VCRs, CD players, and game machines are also becoming smarter and gaining new capabilities. Whether these devices are new or just more powerful versions of existing products, all are becoming increasingly interconnected across a network.

In the six years since Sun's introduction of the Java platform, Java technology has become an essential component for getting work done on the network. With the advent of PersonalJava, EmbeddedJava, and other Java technologies, the benefits of the Java platform are being extended to screen phones, set-top boxes, and even deeply embedded devices. In order to provide compelling Java technology solutions for manufacturers building devices across the spectrum from palmtops to desktops, Sun introduced Java 2 Platform, Micro Edition (J2ME) software.

J2ME is a new, very small Java application environment. It is a framework for the deployment and use of Java technology. Sun will provide J2ME software in configurations suitable for a variety of market segments.

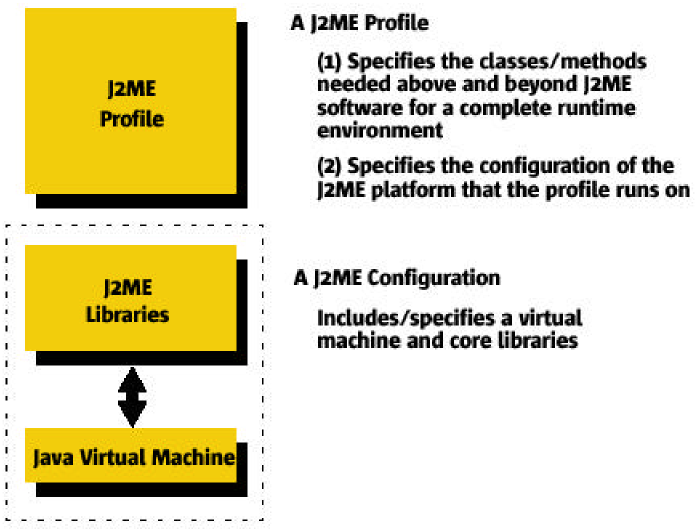

J2ME Profile
A J2ME Profile
(1) Specifies the classes/methods needed above and beyond J2ME software for a complete runtime environment
(2) Specifies the configuration of the J2ME platform that the profile runs on
J2ME Libraries
Java Virtual Machine
A J2ME Configuration
Includes/specifies a virtual machine and core libraries

Figure 18 J2ME configuration

Configuration

A configuration is comprised of a virtual machine, core libraries, classes and APIs. Currently, there are two J2ME configurations: the Connected Limited Device Configuration (CLDC) and the Connected Device Configuration (CDC).

CLDC is designed for devices with constrained CPU and memory resources. Typically, these devices run on either a 16- or 32-bit CPU and have 512 Kbytes or less memory available for the Java platform and applications.

CDC is designed for next- generation devices with more robust resources. Typically, these devices run on a 32-bit CPU and have 2 Mbytes or more memory available for the Java platform and applications.

Built into this core platform is the capability to receive not just application code, but libraries that form part of the Java 2 platform, itself. This enables a J2ME environment to be dynamically configured to provide the environment that the consumer needs to run an application, regardless of whether all the Java technology based libraries necessary to run the application were present on the device when it shipped.

Profile

To further enhance the value of the J2ME environment and assure its ability to provide a focused solution to particular device categories and industries, Sun allows industry groups to define Java technology-based profiles specific to their industry. These profiles are specifications that define a Java technology-based platform suited to a specific industry or class of device.

A profile targeted at the wireless market and utilizing CLDC can retain a very small footprint, consume little power, and provide as much capability as is needed for handheld devices. Devices that require more capable environments can receive profiles that provide additional functionality and are based on CDC. All profiles share a J2ME base as well as the ability to safely download code onto a device and configure the Java environment.

Profiles are defined through the Java Community Process, and may be initiated by industries without Sun's direct involvement. A single company or group of companies can extend tailored Java environments to diverse device types and industries not addressable.

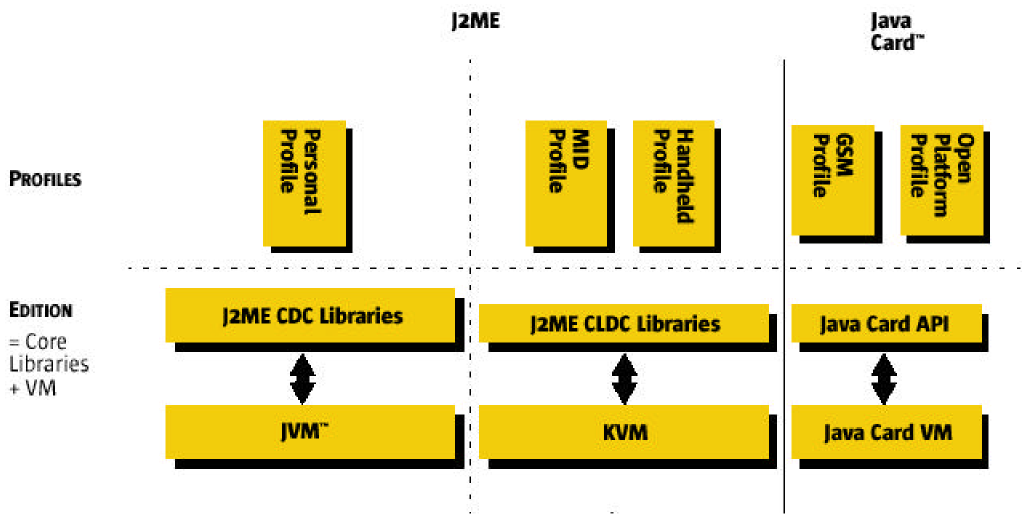


Figure 19 J2ME profile

From PDAs to desktops, the reach of Java technology across the device spectrum and the simplicity of J2ME device-based deployment is key to Sun's strategy of enabling anytime, anywhere service deployment.

10.4 Network Bearers

10.4.1 General Packet Radio Service

The General Packet Radio Service (GPRS) is a wireless data service provided over the GSM network. Figure 17 shows how GPRS (like GSM) is mapped on the lowest layers of OSI protocol stack; therefore, it does not specify the applications running on top of it.

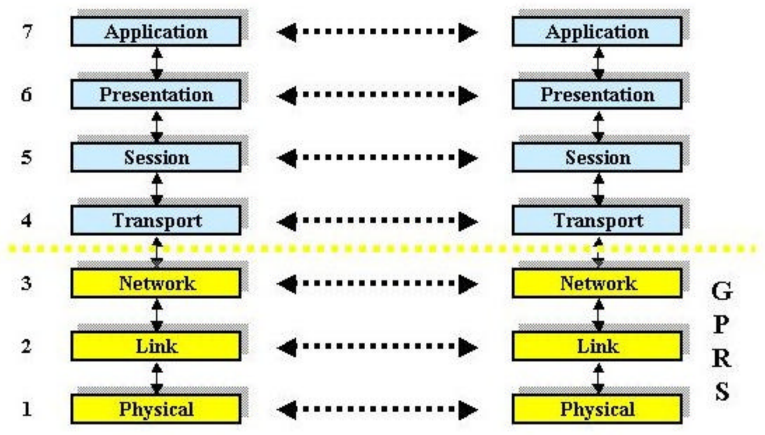


Figure 20 GPRS mapped on the OSI Layers

GSM network adopts a circuit switched system to dedicate a channel for each communication, with 1:1 connectivity between subscribers. Since the channel is dedicated, the billing is based on usage time. When no more channels are available in a cell in a period of time (system overload) the busy signal is supplied for new call attempts. In this connection-oriented paradigm, part of the time is spent in call setup. The dedicated channel offers a low delay, in the order of millisecond; therefore, real-time applications are possible.

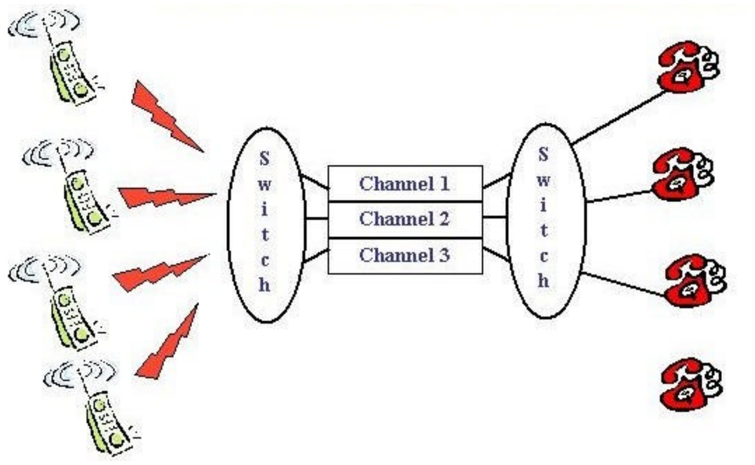


Figure 21 GSM Circuit Switch environment

On the other hand, GPRS allows the sharing of common channels between more users, employing what is called the packet switch system. Each user puts its pieces of information (data packets) on the radio resource, according to the QoS negotiated with the network, and the network switches them to the recipients. Each packet contains the address of the sender and the destination.

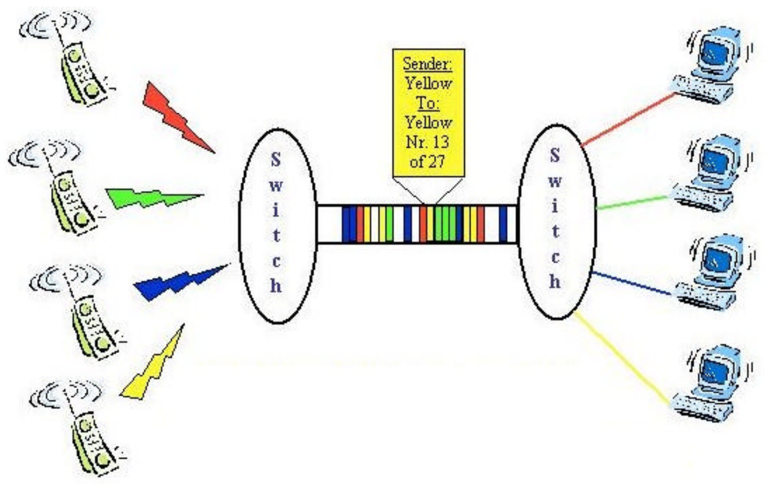
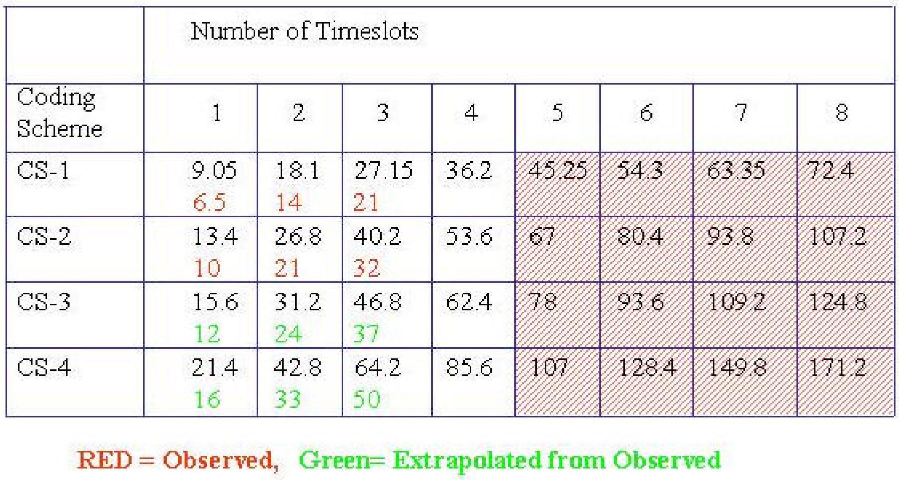


Figure 22 GPRS Packet Switch environment

GPRS packet switch technology introduces several benefits respect to GSM circuit switch: the type of connectivity could be extended also to 1:n and n:m modes; once the user terminal is registered (attached) and the parameters of communication are negotiated, it can send data packets at any time, without no waiting for call setup, and with the ability to have billing based on data amount sent; due to the dynamic use of the radio resource, system overload may slow down the transmissions (data rates, delays) more than rejects new users. Channel sharing, data segmentation, and packet switching introduce network latency (up to 2 seconds) that allows "near real-time" applications more than real-time applications.

Specific GSM radio channels are defined for GPRS, and the allocation of these channels is flexible: from 1 to 8 radio interface timeslots can be allocated per TDMA frame, timeslots are shared by the active users, and up and downlink are allocated separately. The radio interface resources can be shared dynamically between speech and data services as a function of service load and operator preference. Various radio channel coding schemes are specified to allow bit rates from 9 to more than 150 kbps per user.



	Number of Timeslots							
Coding Scheme	1	2	3	4	5	6	7	8
CS-1	9.05 6.5	18.1 14	27.15 21	36.2	45.25	54.3	63.35	72.4
CS-2	13.4 10	26.8 21	40.2 32	53.6	67	80.4	93.8	107.2
CS-3	15.6 12	31.2 24	46.8 37	62.4	78	93.6	109.2	124.8
CS-4	21.4 16	42.8 33	64.2 50	85.6	107	128.4	149.8	171.2

RED = Observed, Green= Extrapolated from Observed

Figure 23 GPRS data rates

As Figure 23 shows, CS-1 coding scheme, which best protects the data in term of residual bit error rate, allows 9.05 kb/s data rate per channel. On the contrary, CS-4, which is used for low quality data transmission, allows 21.4 kb/s data rate per channel.

Three GSM MS modes of operation are supported: An MS in class-A mode of operation operates GPRS and other GSM services simultaneously. An MS in class-B mode of operation monitors control channels for GSM GPRS and other GSM services simultaneously, but can only operate one set of services at one time. An MS in class-C mode of operation exclusively operates GSM services.

User data can be compressed and protected with retransmission protocols for efficiency and reliability. In GSM, GPRS security functionality is equivalent to the existing GSM security. The a network node (SGSN) performs authentication and cipher setting procedures based on the same algorithms, keys, and criteria as in existing GSM. GPRS uses a ciphering algorithm optimised for packet data transmission. A GPRS terminal can access the GPRS services with SIMs that are not GPRS-aware, and with GPRS-aware SIMs. Cell selection may be performed autonomously by a mobile, or the base station system instructs the mobile to select a certain cell. The mobile informs the network when it re-selects another cell or group of cells known as a routeing area.

Once the Packed Data Protocol context is activated (after the mobile GPRS attach), the user terminal is assigned an IP address, that let the user to access the operator Intranet, and the Internet, through a generic gateway or a WAP gateway to access WAP contents.

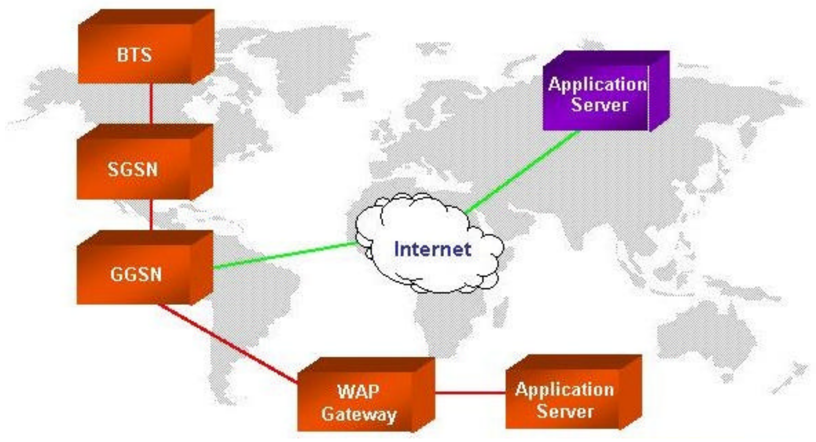


Figure 24 GPRS Internet connectivity

Large volume of data can be exchanged according to the user terminals capability. While technology is moving through devices more powerful, with bigger memory, and an enhanced man-machine interface, it is already possible to employ this reliable, low cost, and fast wireless data transmission via modem connectivity.

Figure 25 GPRS modem session

10.4.2 WAP over GPRS

The Wireless Application Protocol (WAP) is an open, global specification that empowers mobile users with wireless devices to easily access and interact with information and services instantly. The Transport layer protocol in the WAP architecture consists of the Wireless Transaction Protocol (WTP) and the Wireless Datagram Protocol (WDP). The WDP layer operates above the data capable bearer services supported by the various network types, among which GSM, GPRS, and UMTS. As a general datagram service, WDP offers a consistent service to the upper layer protocol (Security, Transaction and Session) of WAP and communicate transparently over one of the available bearer services. The protocols in the WAP family are designed for use over narrowband bearers in wireless telecommunications networks. Since the WDP protocols provide a common interface to the upper layer protocols (Security, Transaction and Session layers), they are able to function independently of the underlying wireless network. This is accomplished by adapting the transport layer to specific features of the underlying bearer.

WAP is designed to work with most wireless networks. Figure 26 shows the mapping of WAP services over the OSI stack, using GPRS as network bearer.

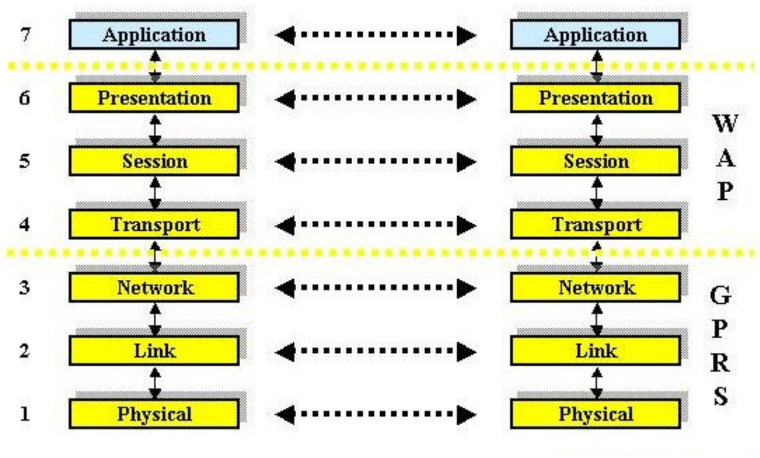


Figure 26 WAP mapped on the OSI Layers

Figure 24 illustrates the protocol profile for the WDP layer when operating over the GPRS bearer service. GPRS supports IP to the mobile therefore UDP/IP will provide datagram services.

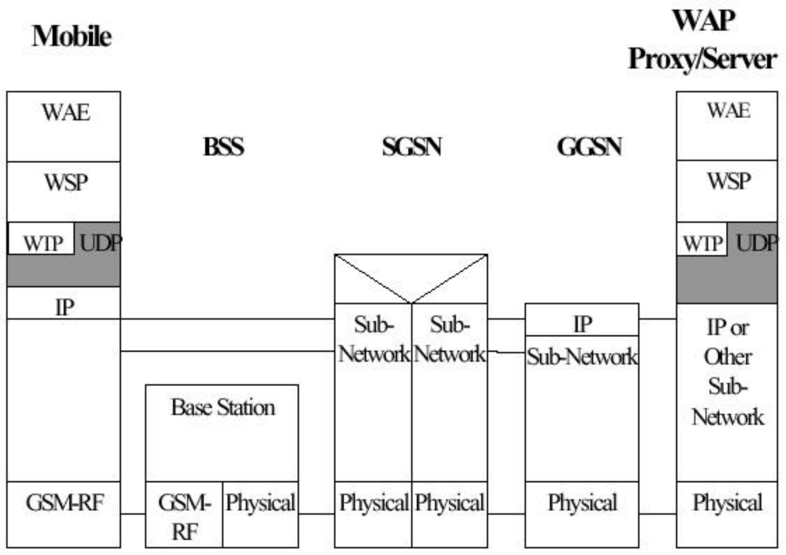
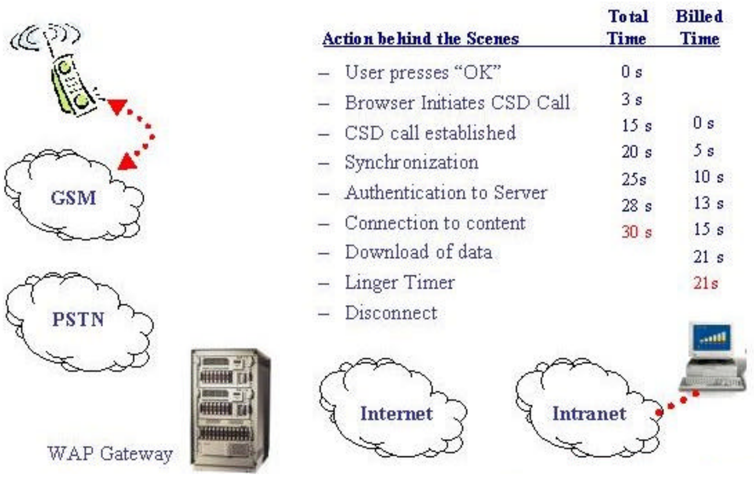


Figure 27 WAP over GPRS

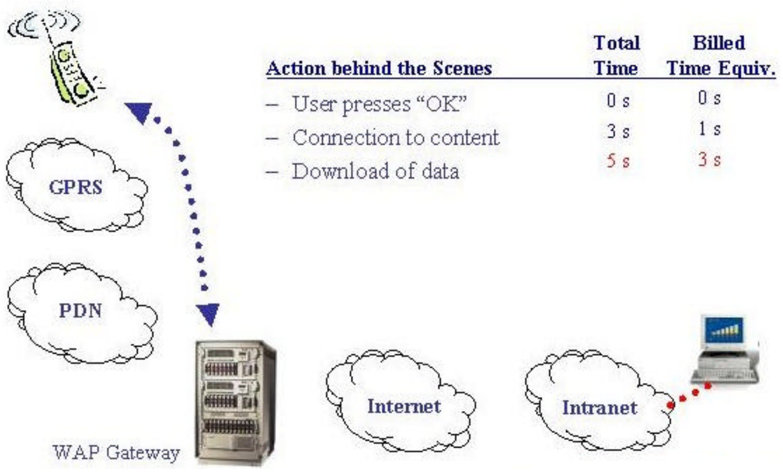
WAP over GPRS introduces a lot of benefits respect to WAP over GSM, in terms of delay and cost. It reduces the wasting of time to establish the connection with the WAP gateway, since a GPRS device is always connected to the IP network, and the costs are based on the actual amount of information exchanged with the content provider.



Action behind the Scenes	Total Time	Billed Time
– User presses "OK"	0 s	
– Browser Initiates CSD Call	3 s	
– CSD call established	15 s	0 s
– Synchronization	20 s	5 s
– Authentication to Server	25s	10 s
– Connection to content	28 s	13 s
– Download of data	30 s	15 s
– Linger Timer		21 s
– Disconnect		21s

Figure 28 Establish a WAP session over GSM

Therefore, WAP over GPRS offers an effective solution for small amount of data exchange, and for request-response data traffic (both interactive and transactional), which are the typical characteristics of Web communications.



Action behind the Scenes	Total Time	Billed Time Equiv.
– User presses "OK"	0 s	0 s
– Connection to content	3 s	1 s
– Download of data	5 s	3 s

Figure 29 Establish a WAP session over GPRS

10.4.3 Universal Mobile Telecommunication System

UMTS (Universal Mobile Telecommunication System) represents a completely different technology respect to GSM, although its backbone relies upon the evolution of GPRS IP infrastructure. It makes a more effective use of the radio resource, with the W-CDMA access technology, offering a higher bandwidth to the subscriber's terminal. Two upper bounds for data rates are foreseen based on user mobility: up to 144 kbps for user with high mobility: up to 2 Mbps for user with low mobility. UMTS channel are placed in the 1900-2200 MHz frequency band.

GPRS is now supporting efficient cost effective packet data on GSM, unleashing the potential of WAP to deliver Internet content on the move. UMTS introduces higher data rates and real time quality of service, allowing both video and voice services to be delivered over the IP path. In the future, we will see a significant step forward with the introduction of IP transport and switching and the emergence of true multimedia services. Continued focus on evolving the standards will see the full realisation of peer-to-peer IP based networks, where services can be fully converged over wire line and wireless IP based access networks. IP based M-Commerce will accelerate and, thanks to more powerful user handsets, applications will run directly on the terminal enabling secure transactions.

10.4.4 WAP over UMTS

The high level architecture for WAP includes the WAP gateway, with the main tasks of encoding/decoding application data, and to terminate the WAP stack at the network end. The encoding reduces the amount of data to be transferred, moreover, it soften the processing constraints of the user terminal. The WAP stack is optimise to work in networks where the radio resource is the most critical, therefore it introduces several optimisations respect to HTTP and TCP.

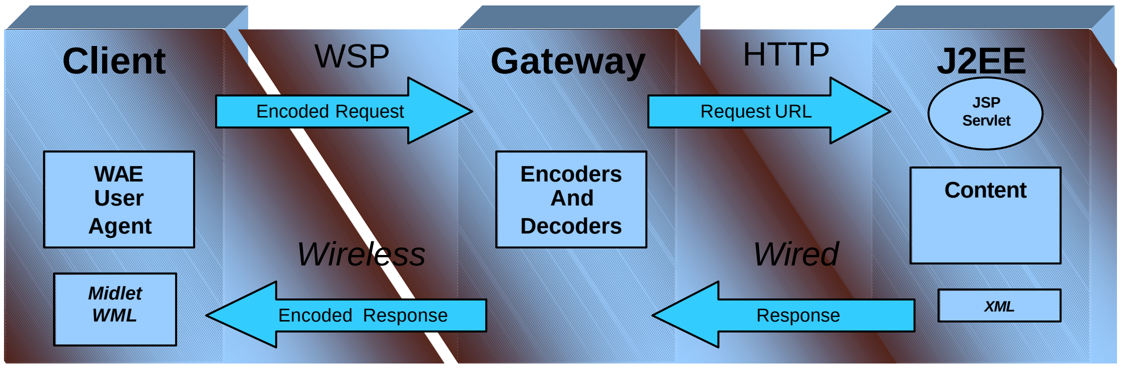


Figure 30 WAP architecture with gateway

In theory, UMTS eliminates all these restrictions in the utilisation of radio resource; therefore, a new paradigm of interaction between the terminal and the network can be taken into account.

Nevertheless, the persistence of handsets limitations, even if only in terms of user interface performances, could justify the use of a sort of gateway, also in the UMTS scenario.

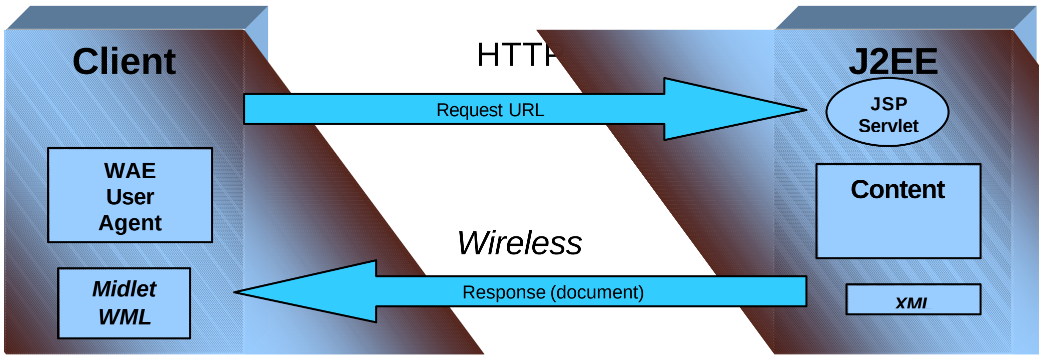


Figure 31 UMTS Internet connectivity without gateway

Figure 30 and Figure 31 represent the above concepts in the context of Java enabled devices.

10.5 Bluetooth

The Bluetooth is specifically designed to provide low-cost, robust, high-capacity ad-hoc voice and data wireless networking in the 2.4 GHz ISM band. Although the band is unlicensed, there is still a number of FCC and other requirements to which adhere. In addition, there are also radio environment problems to address:

- Channel bandwidth is limited to 1 MHz
- Multiple channels, or networks, may not be co-ordinated
- Spectrum spreading must be employed
- Not co-ordinated systems may cause severe interference
- Microwave ovens use this band and can cause interference
- 2.4 GHz IC electronics must run at high current levels

The Bluetooth solution to a robust and low-cost, yet efficient, radio is based on the following characteristics:

- 1 Ms/s symbol rate exploits maximum available channel band width
- Fast frequency hopping avoids interference
- Short data packets maximize capacity during interference
- Fast acknowledgement allows low coding overhead for good links
- CVSD voice coding enables operation at high bit-error rates
- Flexible packet types support wide application range
- Relaxed link budget supports low-cost single-chip integration
- Air interface tailored to minimize current consumption
- Adaptive output power minimizes interference

With these design features, Bluetooth provides extremely flexible and high data rate links in the presence of severe interference. Bluetooth does this without sacrificing performance when signal conditions are good. If the surrounding interference is increased, the degradation is very graceful. Throughout the design of Bluetooth, completely integrated implementation has always been at a premium. The Bluetooth network supports both point-to-point and point-to-multi-point connections. Several ad-hoc piconets (subnets) can be established and linked together. Each piconet is established by a different frequency hopping channel. All users participating on the same piconet are synchronized to this channel. The topology can best be described as a multiple piconet structure.

Bluetooth's technology enables a new set of possibilities and services. Automatic synchronization of a mobile telephone with a desktop/notebook PC or a handheld Personal Digital Assistant (PDA) device will be possible. This could be configured to occur whenever the user comes within range of these devices. It will also be possible to send digital images via a mobile telephone from a Bluetooth-compliant digital camera. Bluetooth will allow a new range of personalized services in which the user is recognized using his PDA or same other Bluetooth device. In an airport lounge room, he could be informed about his ticket, about delays of his fly, about his luggage, and about all the available services in the airport. Users could moreover use the same device to take advantage of all the services provided in the place where they are, to obtain useful information, to locate points of interest and to communicate. As an example, an environment could provide free Internet access, or a messaging system to communicate with the other present people. In a theme park, users could be advised about the beginning of a show, or when their turn comes if they are waiting in a queue for an attraction, and in a museum tourist could obtain explanations and multimedia information about what they are seeing.

10.5.1 WAP over Bluetooth

In many ways, Bluetooth can be used like other wireless networks with regard to WAP. Bluetooth can be used to provide a bearer for transporting data between the WAP Client and its adjacent WAP Server. Additionally, Bluetooth's ad hoc nature provides capabilities that are exploited uniquely by the WAP protocols.

The traditional form of WAP communications involves a client device that communicates with a Server/Proxy device using the WAP protocols. In this case, the Bluetooth medium is expected to provide a bearer service as specified by the WAP architecture.

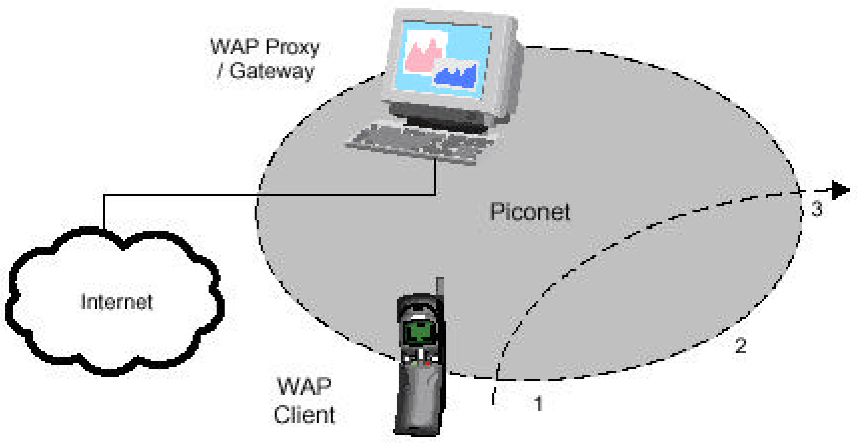


Figure 32 WAP over Bluetooth

11 Ontology system

The Extensible Markup Language (XML) is essentially a meta-language, a language for defining other tag-based languages. This allows individuals and organizations to create tag sets that describe more than just how to display their information. However, this flexibility leads to an interoperability problem: if a university and a furniture store both use the tag <Chair>, do they mean the same or different things? What if another furniture store uses the tag <Seat>? XML DTDs can be used to ensure that a set of documents use the same set of tags, but it would be impossible to create a single DTD that describes everything! As such XML will be very useful as a business-to-business data exchange language and has potential for e-Business, but without something built on top of it, will be insufficient for search. This is where ontology comes in.

11.1 SHOE: Simple HTML Ontology Extensions

11.1.1 Definition

SHOE can be used as the knowledge base language for the implementation of the REGNET ontology checker. SHOE is a small extension to HTML, which allows web page authors to annotate their web documents with machine-readable knowledge. SHOE makes *real* intelligent agent software on the web possible.

HTML was never meant for computer consumption; its function is for displaying data for humans to read. The "knowledge" on a web page is in a human-readable language (usually English), laid out with tables and graphics and frames in ways that we as humans comprehend visually.

The ontology checker can make it possible for web pages to include knowledge that intelligent agents can actually read.

The ontology checker allows n-ary relations, horn clause inference, simple inheritance in the form of classification, multi-valued relations, and a conjunctive knowledge base. It does not currently allow negation, disjunction, or arbitrary functions and predicates.

The ontology checker attempts to make it difficult for entities to pretend to be other entities by providing an easily verifiable key scheme based on URLs.

Agents that will use the ontology system should assume that declarations made by entities are *claims* of those entities, not simple facts. For example, if ten people are claiming to be Marilyn Monroe's lost daughter, a SHOE agent shouldn't be storing the "fact" that Marilyn has ten children.

11.1.2 Compatibility with HTML and XML

A slight variant of the SHOE syntax exists for compatibility with XML. XML is in effect a simplified form of SGML, and thus the XML syntax for SHOE is almost identical to the original syntax. When XML becomes commonplace, the XML variant of SHOE is likely to become the standard.

There are a number of advantages to using XML syntax for SHOE. The XML syntax allows SHOE information to be analysed and processed using the Document Object Model (DOM), thus software that is not SHOE-aware may still use the information in more limited but still powerful ways. Additionally SHOE documents can use the XML standard for style sheets to render SHOE information for human consumption. This is one of the most important aspects because it eliminates the redundancy of having a separate set of tags for the human-readable and machine-readable knowledge.

```
<!-- XML DTD for SHOE -->

<!-- Unlike the SHOE SGML DTD, the XML version does not include the
    HTML DTD. To include XML compliant SHOE and HTML in the same
    document, follow the guidelines for using different namespaces
    expressed in the W3C's "Namespaces in XML" Recommendation. -->
<!-- All element names were changed to lower case to be consistent
    with the definition of XHTML. -->

<!ELEMENT shoe        (ontology | instance)* >

<!-- Since this may be embedded in a document that doesn't have META
    elements, the SHOE version number is included as an attribute
    of the shoe element. -->
```

```
<!ATTLIST shoe
        version         CDATA   #REQUIRED >


<!-- Declarations for ontologies -->
<!ELEMENT ontology     (use-ontology | def-category | def-relation |
                       def-rename | def-inference | def-constant |
                                 def-type)* >


<!ATTLIST ontology
            id              CDATA #REQUIRED
            version         CDATA #REQUIRED
            description     CDATA #IMPLIED
            declarators     CDATA #IMPLIED
            backward-compatible-with        CDATA  #IMPLIED >
```

Figure 33 XML DTD for SHOE

11.1.3 Supported Ontologies

There are a number of SHOE ontologies already developed. These ontologies are open source and they can be used as a starting point for developing the REGNET ontology system. Some of these that can be proved useful to REGNET are listed below:

- Base Ontology, v. 1.0
- Commerce Ontology, v.1.0
- Document Ontology, v. 1.0
- Dublin Core Ontology, v. 1.0
- General Ontology, v. 1.0

11.2 XML TopicMaps (XTM):

11.2.1 Introduction

Topic maps are a new ISO standard for describing knowledge structures and associating them with information resources. As such they constitute an enabling technology for knowledge management. Topic maps are also destined to provide powerful new ways of navigating large and interconnected corpora. While it is possible to represent immensely complex structures using topic maps, the basic concepts of the model – Topics, Associations, and Occurrences (TAO) – are easily grasped. This paper provides a non-technical introduction to these and other concepts (the IFS and BUTS of topic maps), relating them to things that are familiar to all of us from the realms of publishing and information management, and attempting to convey some idea of the uses to which topic maps will be put in the future.

11.2.2 Topics

A topic, in its most generic sense, can be any “thing” whatsoever – a person, an entity, a concept, really anything – regardless of whether it exists or has any other specific characteristics, about which anything whatsoever may be asserted by any means whatsoever.

In fact, this is almost word for word how the topic map standard defines subject, the term used for the real world “thing” that the topic itself stands in for. We might think of a “subject” as corresponding to what Plato called an idea. A topic, on the other hand, is like the shadow that the idea casts on the wall of Plato's cave: It is an object within a topic map that represents a subject. In the words of the standard: “The invisible heart of every topic link is the subject that its author had in mind when it was created. In some sense, a topic reifies a subject...”

Strictly speaking, the term “topic” refers to the element in the topic map document (the topic link) that represents the subject being referred to. However, in this article it is used more loosely to denote both of these things together. Whenever there is a need to distinguish between the two, we use the terms “topic link” and “subject”.

So, in the context of a dictionary of opera, a topic might represent subjects such as “Tosca”, “Madame Butterfly”, “Rome”, “Italy”, the composer “Giacomo Puccini”, or his birthplace, “Lucca”: that is, anything that might have an entry in the dictionary – but also much else besides.

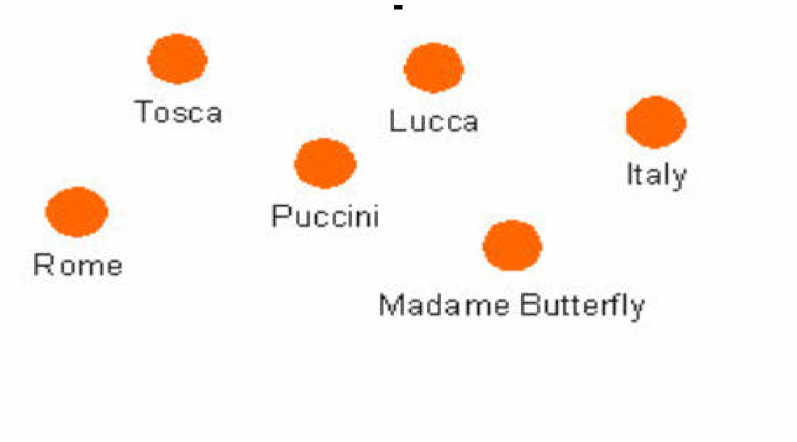


Figure 34 Topics

Topic types

Topics can be categorized according to their kind. In a topic map, any given topic is an instance of zero or more topic types. This corresponds to the categorization inherent in the use of multiple indexes in a book (index of names, index of works, index of places, etc.), and to the use of typographic and other conventions to distinguish different types of topics.

Thus, Puccini would be a topic of type “composer”, Tosca and Madame Butterfly topics of type “opera”, Rome and Lucca topics of type “city”, Italy a topic of type “country”, etc. In other words, topic types represent a typical class-instance relationship.

Exactly what one chooses to regard as topics in any particular application will vary according to the needs of the application, the nature of the information, and the uses to which the topic map will be put: In a thesaurus, topics would represent terms, meanings, and domains; in software documentation they might be functions, variables, objects, and methods; in legal publishing, laws, cases, courts, concepts, and commentators; in technical documentation, components, suppliers, procedures, error conditions, etc.

Topic types are themselves defined as topics by the standard. You must explicitly declare “composer”, “opera”, “city”, etc. as topics in your topic map if you want to use them as types (in which case you will be able to say more about them using the topic map model itself).

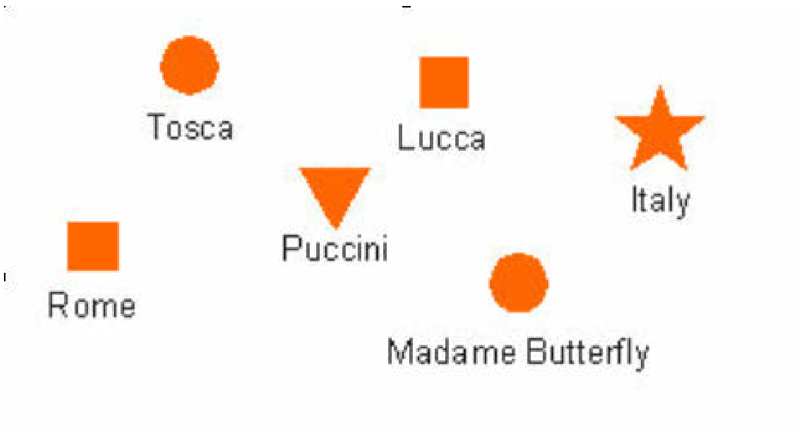


Figure 35 Topic types

Topics have three kinds of characteristics: names, occurrences, and roles in associations.

Topic names

Normally topics have explicit names, since that makes them easier to talk about. However, topics don't always have names: A simple cross reference, such as “see page 97”, is considered to be a link to a topic that has no (explicit) name.

Names exist in all shapes and forms: as formal names, symbolic names, nicknames, pet names, everyday names, login names, etc. The topic map standard doesn't pretend to try to enumerate and cover them all. Instead, it recognizes the need for some forms of name (that have particularly important and universally understood semantics) to be defined in a standardized way, in order for applications to be able to do something meaningful with them, and at the same time the need for complete freedom and extensibility to be able to define application-specific name types.

The standard therefore provides an element form for topic name, which it allows to occur zero or more times for any given topic, and to consist of one or more of the following types of name:

- Base name (required)
- Display name (optional)
- Sort name (optional)

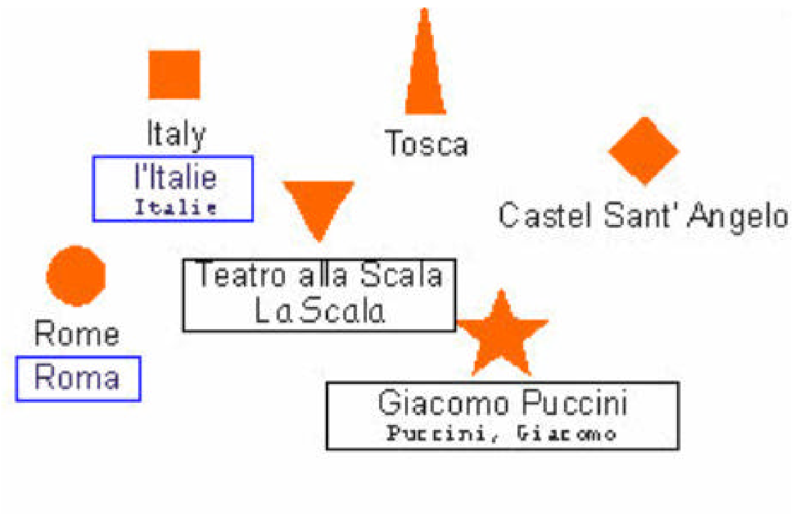


Figure 36 Topic names

The ability to be able to specify more than one topic name can be used to indicate the use of different names in different contexts or scopes (about which more later), such as language, style, domain, geographical area, historical period, etc. A corollary of this feature is the topic naming constraint, which states that no two subjects can have exactly the same name in the same scope.

11.2.3 Occurrences

A topic may be linked to one or more information resources that are deemed to be relevant to the topic in some way. Such resources are called occurrences of the topic.

An occurrence could be a monograph devoted to a particular topic, for example, or an article about the topic in an encyclopaedia; it could be a picture or video depicting the topic, a simple mention of the topic in the context of something else, a commentary on the topic (if the topic were a law, say), or any of a host of other forms in which an information resource might have some relevance to the subject in question.

Such occurrences are generally outside the topic map document itself (although some of them could be inside it), and they are "pointed at" using whatever mechanisms the system supports, typically HyTime addressing or XPointers. Today, most systems for creating handcrafted indexes (as opposed to full text indexes) use some form of embedded mark-up in the document to be indexed. One of the advantages to using topic maps, is that the documents themselves do not have to be touched.

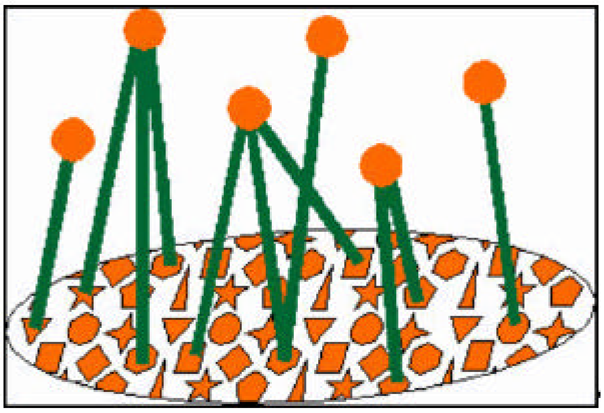


Figure 37- Occurrences

An important point to note here is the separation into two layers of the topics and their occurrences. This separation is one of the clues to the power of topic maps and we shall return to it later.

Occurrence roles

Occurrences, as we have already seen, may be of any number of different types (we gave the examples of "monograph", "article", "illustration", "mention" and "commentary" above). Such distinctions are supported in the standard by the concepts of occurrence role and occurrence role type.

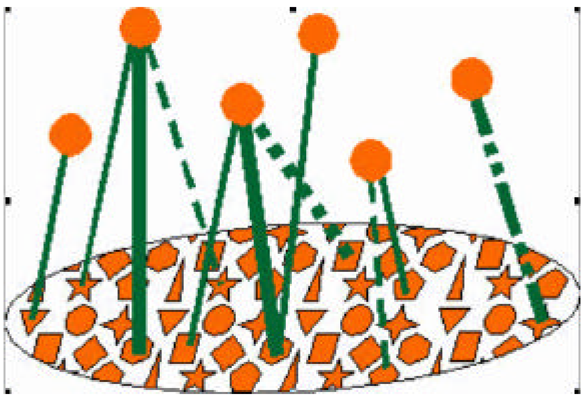


Figure 38 Occurrence roles

The distinction between an occurrence role and its type is subtle but important. In general terms they are both “about” the same thing, namely the way in which the occurrence contributes information to the subject in question (e.g. through being a portrait, an example or a definition). However, the role (indicated by the role attribute) is simply a mnemonic; the type (indicated by the type attribute), on the other hand, is a reference to a topic in the map, which further characterizes the relevance of the role. In general it makes sense to specify the type of the occurrence role, since then the power of topic maps can be used to convey more information about the role.

11.2.4 Associations

A topic association is (formally) a link element that asserts a relationship between two or more topics. Examples might be as follows:

- “Tosca was written by Puccini”
- “Tosca takes place in Rome”
- “Puccini was born in Lucca”
- “Lucca is in Italy”
- “Puccini was influenced by Verdi”

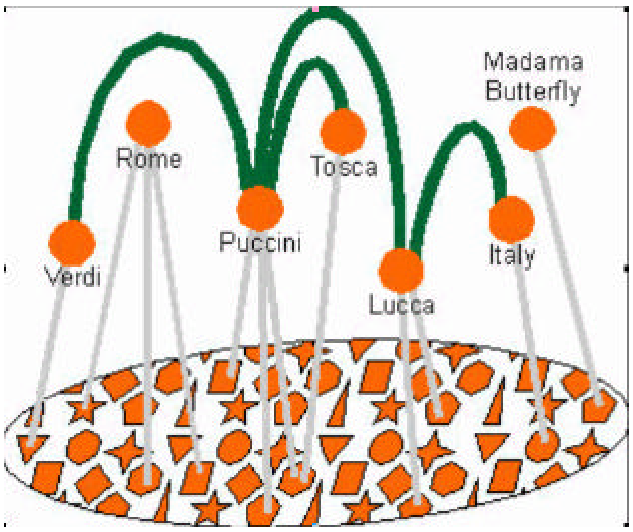


Figure 39 Topic associations

Association types

Just as topics can be grouped according to type (composer, opera, country, etc.) and occurrences according to role (mention, article, commentary, etc.), so too can associations between topics be grouped according to their type. The association type for the relationships mentioned above are written_by, takes_place_in, born_in, is_in (or geographical containment), and influenced_by. As with most other constructs in the topic map standard, association types are themselves defined in terms of topics.

The ability to do typing of topic associations greatly increases the expressive power of the topic map, making it possible to group together the set of topics that have the same relationship to any given topic. This is of great importance in providing intuitive and user-friendly interfaces for navigating large pools of information.

It should be noted that topic types are regarded as a special (i.e. syntactically privileged) kind of association type; the semantics of a topic having a type (for example, of Tosca being an opera) could equally well be expressed through an association (of type “type-instance”) between the topic “opera” and the topic “Tosca”. The reason for having a special construct for this kind of association is the same as the reason for having special constructs for certain kinds of names (indeed, for having a special construct for names at all): The semantics are so general and universal that it is useful to standardize them in order to maximize interoperability between systems that support topic maps.

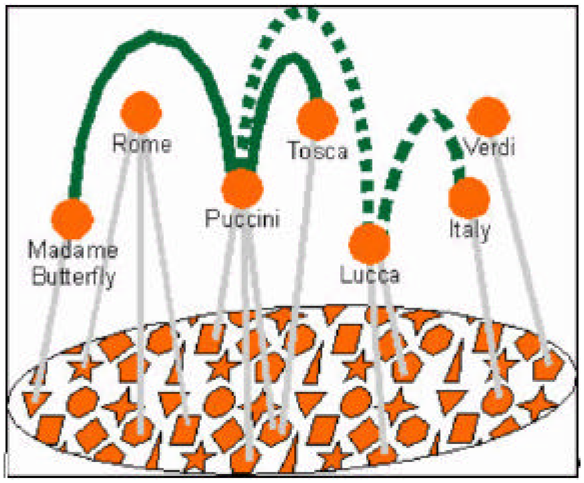


Figure 40 Association types

It is also important to note that while both topic associations and normal cross references are hyperlinks, they are very different creatures: In a cross reference, the anchors (or end points) of the hyperlink occur within the information resources (although the link itself might be outside them); with topic associations, we are talking about links (between topics) that are completely independent of whatever information resources may or may not exist or be considered as occurrences of those topics.

11.2.5 Goals

The design goals for XTM are:

- XTM shall be straightforwardly usable over the Internet.
- XTM shall support a wide variety of applications.
- XTM shall be compatible with XML, XLink, and ISO 13250.
- It shall be easy to write programs that process XTM documents.
- The number of optional features in XTM is to be kept to the absolute minimum, ideally zero.
- XTM documents should be human-legible and reasonably clear.
- The XTM design should be prepared quickly.
- The design of XTM shall be formal and concise.
- XTM documents shall be easy to create.
- Terseness in XTM mark-up is of minimal importance.

12 Themes

The thematic approach of information combination and presentation has always been quite popular in the "physical" world under different forms: articles, papers, music, etc. The grouping of pieces of information into a thematic context and entity is as well challenging for the author as beneficial for the user because there is always a substantial added value in terms of better understanding and situating of the individually addressed topics. But in almost all the cases those theme-based productions are characterised by a high level of uniqueness and rigidity, are severely lacking completeness and do not contain the necessary facilities for keeping the content up to date in an easy way. The physically based thematic solutions are characterised by the existence of quite a gap between the offered possibilities and the ultimate wishes and requirements of the end users.

Things became different with the digital storage of information and the breakthrough of the Internet. Full indexing of information sources and free text search and retrieval mechanisms offered flexibility and a personal touch but generated at the same time noise with a lot of unwanted or irrelevant information. Information providers remedied this by presenting, besides the free text search, a thematic search that offers results that are better focussed on the users' requests. Metadata and harvesters, offering regular updating, came into play and formed with the already classical hypertext and hierarchical tree structures the means to upgrade the thematic approaches. Most of the portals on the web contain now (a combination of) these techniques. Semantic networks are the next hot topic whereby relations (links) between items are characterised by a label and a direction. The latter is fully touching knowledge engineering and is also very promising in the world of personalised education. The new technologies brought solutions to some but not all inadequacies of the thematic approach in the physical world.

Let's focus now on the effect of all this on the world of culture and art. Museums, archives and libraries were long time considered as very conservative, certainly with respect to information technology. But the new technologies reached also this kind of institutions and the first results, mainly by the larger institutions, are more than promising and often quite spectacular and useful. Although there is still a lot of work to do, the digitising of the collections is in full progress and the digitally thematic approach is also entering those institutions. The first initiatives are already available and vary from the classical rigid theme approach of the old days till an automated detection of theme-related objects from a variety of sources. The former contains the old problems and the latter is not mature and stable enough to be put into practice in a very near future. Other problems are the distributed character of theme related objects (located in different institutions) and the lack of accompanying contextual information on the same objects. But the digital age offers also new opportunities: cultural e-Business through fee-based use of multimedia elements and selling physical goods of museum shops via the Internet.

At this stage, a viable solution would be, in a first phase, to manually create a structure of pieces of descriptive information, offering as well freely browsing as combinations with descriptions and images of objects from distributed collections into a theme via relationships and story lines. Those pieces of information should be set up in such a way that they could be reused into other themes and by other authors. Because this structure must be set up almost from scratch, the addition of supplementary characteristics, such as different content and presentation levels is strongly advised together with multilingual facilities. The possibility to create content and metadata at the same time during the editing phase is an enormous advantage. The implementation of these topics could achieve to a large extent the objectives and user requirements mentioned in the previous paragraphs. In further phases, automated harvesting and theme creation techniques could boost the implementation speed and the quantity of theme-based systems.

Another very important aspect of the high level of granularity of the above mentioned descriptive information structure is its ability to play an important role in personalised courseware for learning environments. Prerequisites for a successful implementation for this kind of application are: enough volume and different entry levels.

The typical technologies that should be used to realise these theme-based approaches have to be implemented in a large number of "(sub)-modules" such as: data generation tools, repository management techniques, search and retrieval mechanisms, electronic publishing facilities, reference databases and standards, etc. Referring to the above-mentioned concepts, a lot of existing and emerging technologies and initiatives on data representation, knowledge management and ontology can be considered:

XML/XML Schema	: extended mark up language/schema; platform independent data tagging
JAVA	: platform independent programming language
RDF/RDF Schema	: resource description framework/schema
Topic maps (XTM)	: assignment of different languages, roles, resources to the same concept
Semantic networks	: representation of structured knowledge
(Semantic web)	:
OIL	: ontology inference layer; ontology representation and inference language

ADL – SCORM : sharable content object reference model; personalised courses

Some of these technologies are quite new, not always on the conceptual side but certainly on the practical implementation side. Others lack stability and standardisation. This can easily be detected on the respective articles treating these issues on the Internet and in the professional literature.

The definition of themes - restrictions and priority

There are many ways of presenting existing themes. Putting them in a fixed hierarchical structure cannot embrace all different aspects, typical for it. This feature is structurally restricting and creating any kind of classification endures severe critics of the specialists of separated profiles. The determination of any frame of themes contains restrictions and rigidity because a theme could be seen in different aspects and it could be put in more than one topic of the determined frame.

One of the original approaches could be the use of PIECES OF INFORMATION (FRAGMENTS) related with the themes and digitised objects. Fragments give a green light for flexibility and adaptability of the system. In such a way it is possible that one object can be related to different themes. Moreover the preliminary definition of some themes could not be considered as a restriction of a future extension of the themes' area.

But the more common definition of themes offers the customer a wider entry field and by, later on, narrowing the objects of interest he could easier surf the topics. The more common description of themes is more convenient not only for the user who want to see what the system looks like, but for the user who will be the business consumer, who would like to use the business possibilities of system or would like to include new themes (by Data Generation sub-system).

For instance – let's suppose that the consumer is Historical Museum. It has seen the topic HABSBURGS, and decide to become a user and to present its objects (for instance MEDICI) in REGNET. If the theme is more common, for example HISTORICAL HERITAGE the user will choose it and will propose his topic "Medici" to be included in the theme HISTORICAL HERITAGE.

In other words the aggregation of the thematic is not so needed to the content providers because they will present their objects in system, but it is suitable for the consumer and represents the so called "User Friendly Interface" for the system.

The more common definition of themes or a hierarchical structure of themes is preferable for the designer of the Website because he can arrange easily all themes that are relevant.

It is important that these more common themes are included in the beginning of the digitising process of the system because they use fields which are about relations with other themes or topics. Using the *Relation Element* of Dublin Core or *Key words* of the *Fragments*, which have to be filled for all objects, it could be beneficial to begin with more common terms and narrow the terminology.

As the process of determining the themes is not restricted in time, existing schemes could not be considered as fixed but later on they can be changed for the better.

13 Acronyms

Here are main acronyms used in this document.

ADO	ActiveX Data Object
API	Application Program Interface
CGI	Common Gateway Interface
CORBA	Common Object Request Broker Architecture
EAI	Enterprise Application Integration
ECMA	European Computer Manufacturers Association
EJB	Enterprise JavaBeans
GPRS	General Packet Radio Service
GUI	Graphical User Interface

HDML	Handheld Device Markup Language
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
HTTPS	Hyper Text Transfer Protocol over SSL
IDL	Interface Description Language
IIOP	Internet Inter ORB Protocol
J2EE	Java 2 Enterprise Edition
J2ME	Java 2 Micro Edition
J2SE	Java 2 Standard Edition
JDBC	Java DataBase Connectivity
JMS	Java Messaging Service
JNDI	Java Naming and Directory Interface
JRMP	Java Remote Method Procedure call
JSP	Java Server Pages
JTA	Java Transaction API
JTS	Java Transaction Service
JVM	Java Virtual Machine
LDAP	Lightweight Directory Access Protocol
MOM	Message Oriented Middleware
ORB	Object Request Broker
OTM	Object Transactional Monitor
PDA	Personal Data Assistant
PDF	Portable Document Format
PSTN	Public Switched Telephone Network
RMI	Remote Method Invocation
SOAP	Simple Object Access Protocol
SSL	Secure Socket Layer
UMTS	Universal Mobile Telecommunication System
URL	Uniform Resource Locator
WAE	WAP Application Environment
WAP	Wireless Application Protocol
WML	Wireless Markup Language
WSDL	Web Service Description Language
WSP	Wireless Session Protocol
WTSL	Wireless Transport Security Layer
XML	EXtensible Markup Language
XSL	Extensible Stylesheet Language

XSLT	XSL Transformations

14 Protocol Z39.50

In order to make all databases accessible from a single interface, they are all connected to the Internet through Z39.50. This is a standard client/server search and retrieve protocol. Since it was originally proposed in 1984 it has increased in popularity in the library community, becoming one of the most used standards to achieve interoperability in search and retrieval.

As Z39.50 is supposed to cover all possible needs for searches in bibliographic databases, its design is quite flexible, allowing the administrator to choose which features of the protocol to implement and how to use these to achieve the desired functionality.

In order to make it easier to create software that can work with several databases, some sub-standards to Z39.50 have been developed. These standards determine what features a Z39.50 server should support and how it should respond to certain requests.

The gateway approach, described on CIMI Profile (including Z39.50 protocol) provides the capability to convert dynamically between the Z39.50 protocol and the native query and retrieval facilities of the host database. In other word it maps the host database's searching and retrieval capability on to the Z39.50 Profile as well as it can, rather than augmenting it in any fundamental way. The "underlying database" is accessible according to the mapping implemented by the Z39.50 protocol.

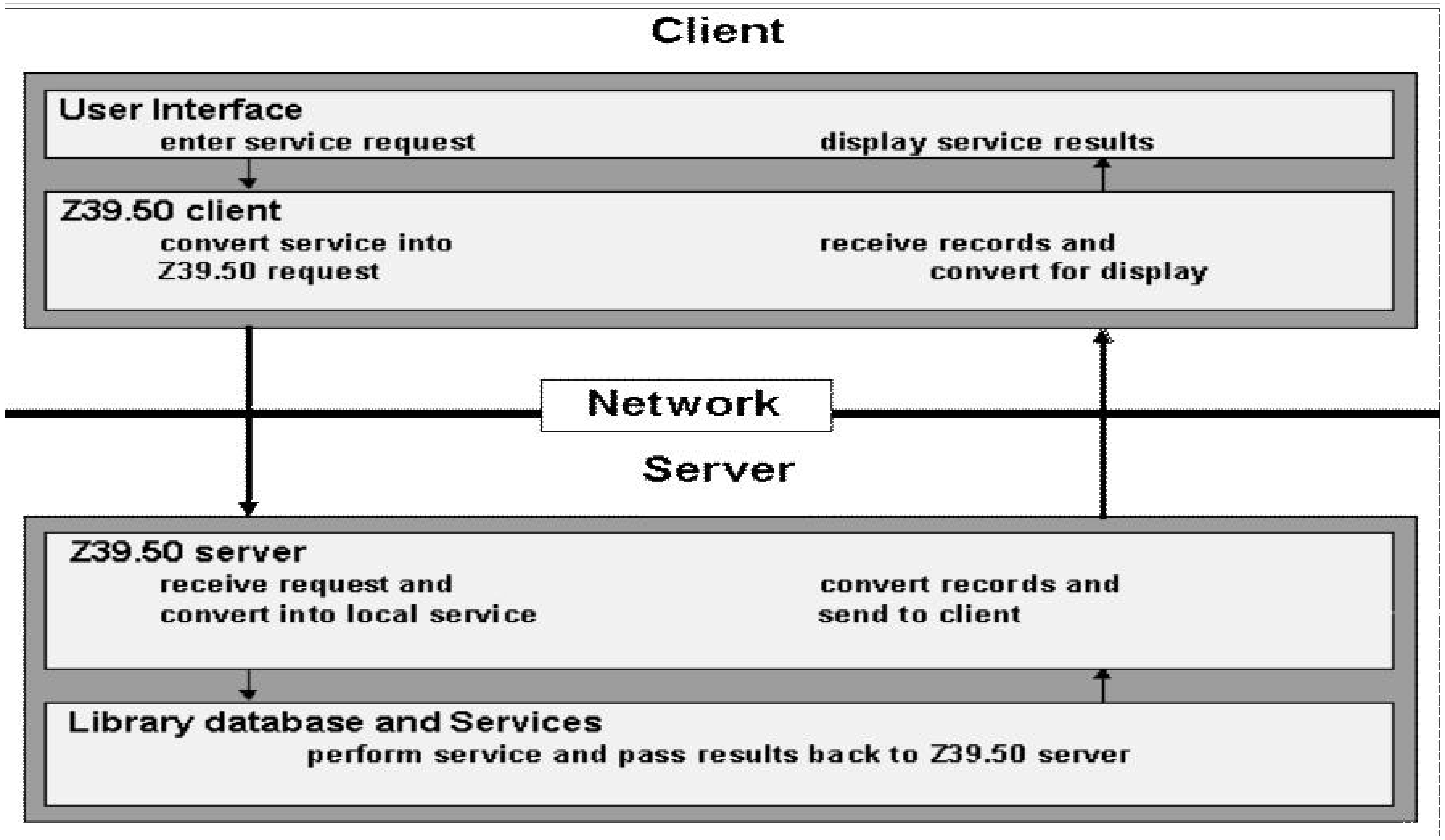


Figure 41 Protocol Z39.50

List of Figures