

UNIVERSITÀ CA' FOSCARI – VENEZIA  
Corso di Laurea magistrale (ordinamento ex D.M. 270/2004)  
in Informatica - Computer Science  
a.a. 2012-2013

Tesi di Laurea

Enterprise Distributed Systems:  
the case of the Sybase Unwired Platform

Laureando: Daniele De Rosa  
Matricola: 813917

Relatore: Chiar.mo prof. Renzo Orsini

# Contents

<b>1</b>	<b>Abstract</b>	<b>1</b>
<b>2</b>	<b>Introduction</b>	<b>3</b>
<b>3</b>	<b>Distributed Systems</b>	<b>7</b>
3.1	Transparencies . . . . .	7
3.2	Middleware . . . . .	8
3.3	Architectural models . . . . .	10
3.3.1	Client-Server model . . . . .	10
3.3.2	Client-Server model with multiple server . . . . .	10
3.3.3	Peer-to-peer model . . . . .	11
3.3.4	Proxy Server model . . . . .	12
3.4	Architectural patterns . . . . .	13
3.4.1	Layering . . . . .	13
3.4.2	Tiered architecture . . . . .	14
3.5	Addressing . . . . .	16
3.5.1	LAN . . . . .	16
3.5.2	Broadcast . . . . .	16
3.5.3	Name Server . . . . .	16
3.6	Remote Invocation . . . . .	17
3.6.1	Request-reply protocols . . . . .	17
3.6.2	Interfaces . . . . .	19
3.6.3	Remote Procedure Call (RPC) . . . . .	20
3.6.4	Remote Method Invocation(RMI) . . . . .	21
3.6.5	Events . . . . .	22

3.7	Processes' scheduling . . . . .	22
3.8	Mutual Exclusion . . . . .	23
3.8.1	The central server algorithm . . . . .	23
3.8.2	Distributed algorithm . . . . .	24
3.8.3	Ring-based algorithm . . . . .	25
3.9	Time . . . . .	26
3.9.1	Cristian algorithm . . . . .	27
3.9.2	Berkeley algorithm . . . . .	27
3.9.3	Network Time Protocol (NTP) . . . . .	27
3.10	Replication . . . . .	28
<b>4</b>	<b>Sybase Unwired Platform</b>	<b>31</b>
4.1	Platform Solution . . . . .	32
4.1.1	Sybase Unwired Platform Runtime . . . . .	32
4.2	Cluster and Non-cluster systems . . . . .	36
4.2.1	Cluster Types . . . . .	38
4.3	Mobile Business Object (MBO) . . . . .	41
4.4	Data Synchronization and Data refresh . . . . .	48
4.4.1	Unwired Server Cache . . . . .	49
4.4.2	Data Refresh . . . . .	50
4.4.3	Cache Refresh schedules . . . . .	52
4.4.4	Data Change Notification (DCN) . . . . .	55
4.4.5	Synchronization . . . . .	58
<b>5</b>	<b>Implementation of an HTML5/JS Hybrid Application</b>	<b>63</b>
5.1	The purpose of the application. . . . .	65
5.2	Implementation of Mobile Business Object . . . . .	65
5.2.1	Data Source and Profile Connection . . . . .	66
5.2.2	Method Definition . . . . .	67
5.2.3	Load Arguments . . . . .	69
5.2.4	Relationship . . . . .	70
5.3	Cache Group and Synchronization Group . . . . .	72
5.4	Workflow . . . . .	75
5.4.1	Code HTML5 and Javascript . . . . .	75
5.4.2	Online Request . . . . .	80





# List of Figures

3.1	Client-server . . . . .	10
3.2	Client-server with multiple server . . . . .	11
3.3	Proxy server . . . . .	12
3.4	Layering . . . . .	13
3.5	Two-tier solution . . . . .	15
3.6	Three-tier solution . . . . .	15
3.7	Name server . . . . .	17
3.8	Request-reply protocols . . . . .	18
3.9	Remote Method Invocation . . . . .	21
3.10	Remote Method Invocation . . . . .	22
3.11	Central Server algorithm . . . . .	24
3.12	Ring algorithm . . . . .	25
3.13	Replication system . . . . .	29
3.14	Passive Replication . . . . .	30
3.15	Active Replication . . . . .	30
4.1	Sybase Unwired Platform . . . . .	31
4.2	Sybase unwired Platform Runtime . . . . .	33
4.3	Relay Server . . . . .	35
4.4	Non-clustered systems . . . . .	37
4.5	Unwired Platform servers in clusters . . . . .	38
4.6	Unwired Server cluster . . . . .	39
4.7	Unwired Server connection to Relay Servers in cluster . . . . .	40
4.8	Mobile Business Object . . . . .	42
4.9	Mobilize EIS data . . . . .	43
4.10	Bind attributes of two MBO . . . . .	45

4.11	Relationship between MBO . . . . .	46
4.12	Data Synchronization and Data Refresh . . . . .	48
4.13	Data Refresh . . . . .	51
4.14	Data Refresh Initiated by Unwired Server . . . . .	53
4.15	Data Change Notification . . . . .	55
4.16	Data Change Notification without payload . . . . .	56
4.17	Data Change Notification with payload . . . . .	57
4.18	Synchronization . . . . .	58
4.19	Combine Synchronization and Data Refresh strategies . . .	60
4.20	Synchronization initiated by Unwired Server . . . . .	61
5.1	Hybrid Web Container . . . . .	64
5.2	Possible Data Source . . . . .	66
5.3	Connection Profile . . . . .	67
5.4	Method definition . . . . .	67
5.5	Attributes mapping . . . . .	68
5.6	MBO of the output tables . . . . .	69
5.7	Search person BP screen . . . . .	69
5.8	Personalization Key . . . . .	70
5.9	Load Arguments . . . . .	70
5.10	Relationships . . . . .	71
5.11	One-to-many and composite . . . . .	71
5.12	Relationship between MBO . . . . .	72
5.13	Cache Group Policies . . . . .	72
5.14	Cache Group . . . . .	73
5.15	Cache Group divided . . . . .	74
5.16	Synchronization . . . . .	74
5.17	Synchronization interval . . . . .	74
5.18	Start page . . . . .	75
5.19	Pop-up confirmation window closing . . . . .	80
5.20	Search person BP screen . . . . .	84
5.21	Online Request . . . . .	84
5.22	Personalization Key mapping . . . . .	85
5.23	List of the person BP with surname Monti . . . . .	85
5.24	Details of person BP Ambrogio Monti . . . . .	86

# Chapter 1

## Abstract

Analysis of a distributed system, open problems and challenges. Description the distributed system Sybase Unwired Platform (SUP) and how it solves such problems. Implementation of a mobile application hybrid through the SUP.





# Chapter 2

## Introduction

In this paper we will analyze two systems that are spreading more and more in our everyday lives. One is more visible, and it is the mobile system, the other is the distributed system.

The advent of smartphones, and the continuous growing of their computing power, has meant that they can often replace the normal computer, with the significant advantage of being always handy, in the true sense of the word. They can now in fact perform various activities, from those working to those of leisure.

The services available are very often used on our devices thanks to the implementation of distributed systems: they in fact allow the collaboration of several calculation units, independent of each other, for the achievement of the intended purpose, for example the use of a mobile application.

Distributed systems, however, present many challenges to their implementation and use, due to the diversity of the devices involved, the variety of communication protocols to connect them to each other, for the different platforms and programming languages used by each of them. It also added the difficulties related to the scalability of the system, without reducing the performance and consistency of the data. A number of issues, as can be noted, which must from time to time find answer.

I had the opportunity to meet with all of these issues when I did my

internship, scheduled for my university curriculum.

The company at which I had the opportunity to do so, having decided to transfer part of its services on mobile had to lean precisely to a distributed system. Critical component is the Sybase Unwired Platform (SUP): my job, during the training period, was to create a hybrid application capable of being able to run on any mobile platform (Android, iOS, Windows Phone, BlackBerry), it can connect to CRM test environment of Bocconi University and then transfer some of the services offered by the CRM server on mobile devices via SUP.

Therefore, it is through this experience that I had the opportunity to be able to see up close a distributed system, its implementation and its use, clashing with its problems and possible solutions.

The application that I was going to make would have to read the personal data of a Business Partner (BP) - person or company that was - such as full name, company name, relationships with other BP, etc., it would allow the user to set activities performed or to be performed by the BP in the CRM environment testing, such a course that a teacher would have to hold, and finally, would have to read the leads relative to a BP and to be able to change their state (where for lead means an event that happened to BP, for example, a donation).

The question that one is entitled to ask, and that I did in starting my job, is: why having to go through the Sybase Unwired Platform to communicate with the server CRM Bocconi, instead of allowing direct communication of mobile devices with these servers?

Driven by this curiosity, I started studying the documentation provided by SAP for understanding the behaviour of the Sybase Unwired Platform, its interaction with the SAP server and mobile devices. So I found out what were the advantages of this method: the SUP together with the CRM server and the mobile devices goes to form a distributed system, whose calculation unit collaborate and communicate with each other, allowing a user to be able to read, create, and modify data within the CRM test environment of Bocconi, simply using their smartphone, and the reason because the SUP is used to communicate with the CRM server is

that it solves all those problems and transparencies which must resolve and provide a distributed system.

It became immediately evident as the SUP was going to solve the problem of scalability, since potential mobile devices that communicate with the CRM grow in number over time, and the multiple interactions through direct communication may be clogging it. Not to mention that an increase in the workload of the CRM server would reduce the performance of the entire distributed system: however the SUP offering, for example, caching capacity, avoids equal requests to the server, allowing performance transparency, even in case of increase of workload. The caching also enables a fault transparency for the CRM server, so that, even if the server is dropped, however, allow mobile devices to continue to communicate with the SUP. The latter allows, again, the competition among the multitude of mobile devices that operate on shared resources without interfering with each other. In conclusion, the SUP may be installed as a cluster system, in order to allow the replication of the data, increasing the availability of the resources and increasing the overall performance of the system. Not to mention that the replication helps to provide many of the transparencies already discussed, such as scalability and fault tolerance.

The heart of the SUP is its middleware layer, called Mobile Business Object (MBO), which provides another set of transparencies, such as location: mobile devices has no knowledge of where the CRM server is, but for them is sufficient connect to the SUP, easily accessible via a proxy server, called Relay Server. The MBO also provides two other important transparency: the platform and programming languages. The SUP in fact not only allows you to communicate to a server with the SAP platform, but also with different types of Web Server or Database: the MBO is that interface that hides the different types of platforms and gives a common vision.

To implement an application for each type of mobile platform (Android, iOS, BlackBerry, Windows Phone) was no longer a risk, because the SUP provides a development environment that can be able to create hy-

brid applications - that is written in HTML5, Javascript and CSS - and, thanks to a web engine, to be used as Web Applications that run inside the smartphone and communicate with the SUP. The web engine made available by SAP is a container, written in the native language of the platform on which it runs, and it is provided of libraries that allow the communication with the SUP. The hybrid application is loaded into the container that takes care of translating the HTML5, Javascript and CSS code in the native code of the platform in question. The container stand for the middleware layer within the mobile device providing platform and programming language transparency. Then the advantages that a hybrid application provides are evident: its ease of development (HTML5, Javascript and CSS code instead of native code) and the ability to resolve a fundamental question, namely to write application code once, without having to repeat this with the native code for each mobile platform used.

With this document I will go to through all the problems since here briefly illustrated, first through the analysis of a distributed system in general, and then descibing how they have been addressed and resolved in a particular distributed system, the SUP. Finally I will show how a hybrid application was implemented, discussing its capability to solve problems discussed.

# Chapter 3

## Distributed Systems

A distributed system is a system consisting of multiple autonomous processing units that support a set of processes and/or data bases and that interact cooperating to achieve a given objective. The processes coordinate and exchange information via a communication network and passing messages.

### 3.1 Transparencies

The challenges and problems that a distributed system is facing are:

- ***Concurrency transparency***: allows a set of processes to operate concurrently sharing resources and without interference between them. The operations that work on the shared data must be synchronized, so as to maintain the data in a consistent state.
- ***Replication transparency***: allows the use of multiple copies of resources to increase the reliability and performance, without the users becoming aware of them.
- ***Fault transparency***: allows the masking to faults so that users can complete the requested operations even in the presence of faults of hardware and software components.

- ***Mobility transparency***: allows you to move resources and customers in a system without affecting the operations of user.
- ***Performance transparency***: allows to reconfigure the system to vary the load to improve the performance.
- ***Scalability transparency***: if a system is *scalable* then the system and applications expands in a scalable manner, that is there is a significant increase in the number of resources and the number of users, without changing the structure of the system or application algorithms. The problem of scalability is a recurring theme in the development of distributed systems. The main techniques that successfully address this problem are: data replication, caching technique, deployment of multiple server and the use of processes that work concurrently.

## 3.2 Middleware

The middleware layer using protocols based on message passing between processes to provide a high level of abstraction as a remote invocations and events. An important aspect of the middleware is the provision of transparency and independence from the details of underlying communication protocols, operating systems, hardware and programming languages. In addition to solving the problems of heterogeneity, middleware provides a uniform computational model to use by the programmers of servers and distributed applications. It offers the following types of transparency:

- ***Location transparency***: for example with the communication protocol Remote Procedure Call (RPC), the client calls a procedure that can not distinguish whether it runs in the same process or a different process, possibly on a different PC. Neither the client needs to know the location of the server. Similarly, with the communication protocol Remote Method Invocation (RMI), the objects done invocations can not distinguish whether the invoked object is local or not, and without knowing its location.

- ***Protocol transparency:*** The protocols that support the middleware are independent from underlying transport protocols.
- ***Platform transparency:*** Hides the difference between two hardware and software architectures.
- ***Programming languages transparency:*** Some middleware are designed to allow applications of distributed systems to utilize more than one programming language. Such as Common Object Request Broker (CORBA) allows clients written in a programming language to invoke methods in objects that live in server programs written in other languages.



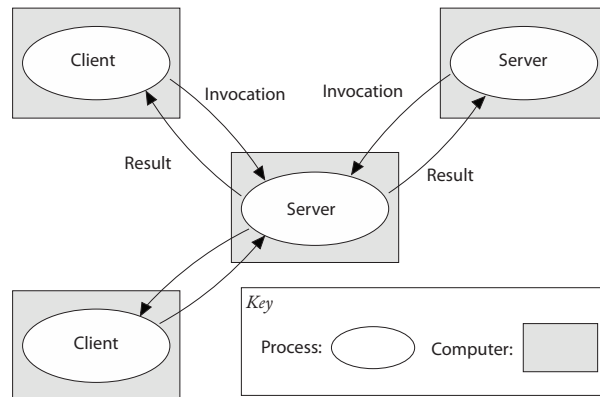
### 3.3 Architectural models

A distributed system can take different architectural models. The model is chosen according to the role played by the various processes within the distributed system to achieve the purpose assigned to it.

#### 3.3.1 Client-Server model

Client processes require shared resources and shared services to a server process. Client and server processes runs almost always in a different hosts as well as the client can run on different hosts. In the client server model most of the work load is assigned to the process server.

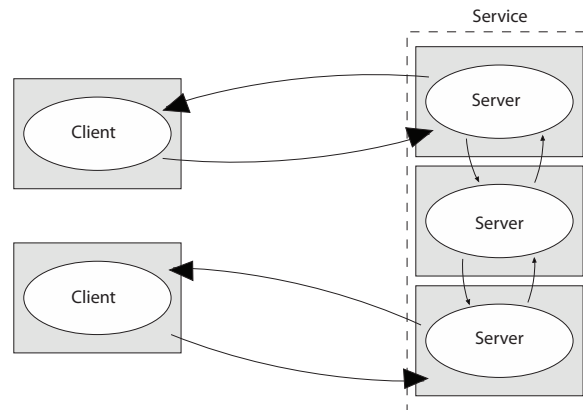
Figure 3.1: Client-server



#### 3.3.2 Client-Server model with multiple server

It is a client-server model in which there are multiple host servers that interact with each other to offer their services to clients. The resources and services are divided among the different server or replicated across multiple servers. This model is designed to not award the entire workload to a single server.

Figure 3.2: Client-server with multiple server



### 3.3.3 Peer-to-peer model

In this architectural model, all the processes involved play similar roles, that is run the same program, in practical they interact with each other without distinction between client and server processes. While the client-server model is a centralized type model, where services and resources are shared among the clients, served by the server, in peer-to-peer each peer is able to provide resources and services.

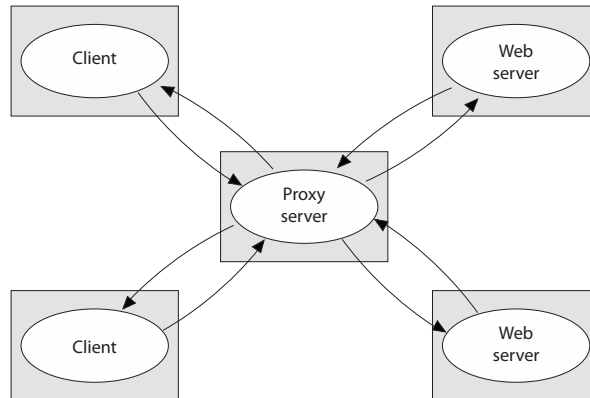
This model is driven by the need to distribute resources between the host in such a way to share the computing and communication loads due to the increasing number of clients that access to shared resources. In this model the network and computing resources owned by the user using the service can be made available also to offer such service.

Essentially the purpose of a peer-to-peer model is to achieve the fulfilment of a given task through the use of the resources held between the various peer and not from a single host server, in order to spread the workload among all participants.

### 3.3.4 Proxy Server model

A proxy server or reverse server is a server process intermediary between the clients and the server. Clients connect directly to the proxy server instead of the real server with the aim of increasing the availability and performance of the service by reducing the load of the servers, in the figure below Web Server.

Figure 3.3: Proxy server



The response of the real server is sent to the proxy server which in turn sends to the clients, so the clients are not aware of the address of the real client.

The installation of a proxy server leads to many advantages. Including caching, namely the use of a cache which stores the information used most recently. When you receive a resource from the server it is stored in the cache by deleting or updating existing information in the cache. When a client requests a resource, it checks if it is already present in the cache, otherwise you have to make a new request to the server. The cache can then be positioned within a proxy server so as to be shared between the different clients. Another advantage of the proxy server is load balancing by distributing the load across servers. Moreover the proxy server is internet facing providing access only to certain services,

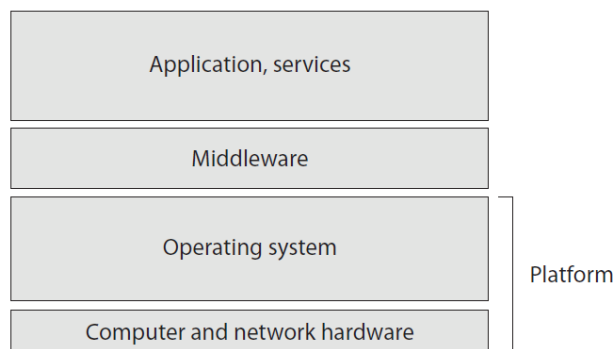
protecting server from attacks.

## 3.4 Architectural patterns

### 3.4.1 Layering

A system divided into different layers, each of which provides services to the next layer. This architectural pattern provides an abstraction layer between the various layers as a layer does not know the implementation of the underlying layers, but is able to use the services offered by them. For a distributed system, the subdivision of the layers can be represented by the figure:

Figure 3.4: Layering



The layers “Operating Systems and Computer and Network hardware” represent together the platform on which the distributed system runs. They offer services to the higher layers that can be implemented regardless of how it is made up the underlying platform: different operating systems and communication protocols.

While middleware consists of processes and objects that interact with each other to provide services for communication and sharing of resources to distributed applications.

### 3.4.2 Tiered architecture

The layering takes care of the vertical organization of services, while the tiering is responsible for organizing the functionality within each layer, positioning each functionality in a given server.

There are two types of tiering: two-tiered and three-tiered. Before going into details of both types of architectures, we divide an application into three functional parts. The first concerns the *presentation logic*, that is how the application is displayed to the user allowing him to interact with it. Then it regards how to display views and how that views should be updated depending on the user interactions. The second concerns the *application logic*, that is goes into more detail on how the application is implemented, is also called *business logic*. The third concerns the *data logic*, tha is the storage data used by the application, tipicaly a database management system.

#### Two-tier solution

The three functional parts of an application, mentioned before, are divided between two processes, the client process and the server. This subdivision is achieved by the division of the *application logic* into two parts, one located in the client and the other in the server. The advantage of this type of architecture is the speed with which the interaction can take place as an operation may be invoked by the exchange of a single message. The disadvantage is due to the fact that the application logic is divided into two parts and then it can not always be a direct invocation of a part of the logic with another.

#### Three-tier solution

Through this solution, each of the three functional parts of an application is associated with a physical server. So each tier has a more specific role, for example, the first tier may be a graphical user interface, while the third tier a database.

The disadvantage is due to the management of three server, and then the increase of messages to each operation.

Figure 3.5: Two-tier solution

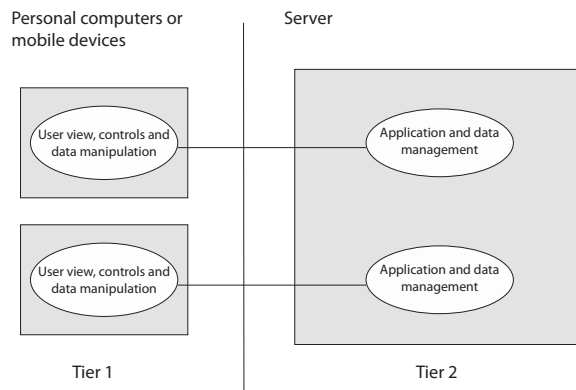
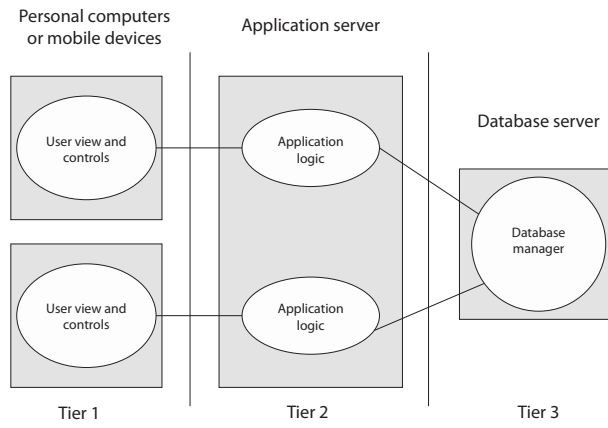


Figure 3.6: Three-tier solution



## **3.5 Addressing**

Addressing means the localization of the servers from the processes that need to request services.

### **3.5.1 LAN**

Requests shall be sent to the physical address, it implies that the client knows the physical address of the servant, so it lacks the transparency of the location. Also not convenient when you need to reallocate servant because as result you have to change the code of the client.

### **3.5.2 Broadcast**

To solve the problem of reallocation of servant, is sent via broadcast a special package of localization and servant responds with a message "I'm here!", with its own physical address. Then the client sends the request to that address and the servant sends a reply to the client.

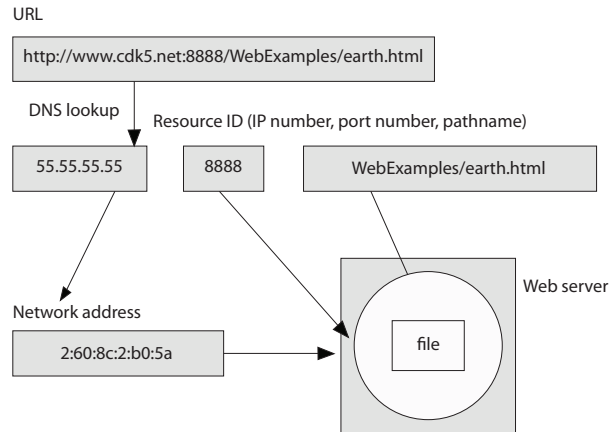
### **3.5.3 Name Server**

We can see that with the broadcasting we have the disadvantage of the considerable network traffic. A more effective method is the use of a Name Server which knows addresses of all server. The Name Server maintains the mapping between the logical names and physical addresses of the servers.

The client sends a request to the Name Server to obtain the address of server. The Name Server responds with a reply containing the physical address of the server. Now the client can send the request to the server and the latter respond to the client.

The Name Server is a centralized component may therefore be appropriate to replicate it.

Figure 3.7: Name server



## 3.6 Remote Invocation

Now we will discuss the main techniques of communication between processes of a distributed system.

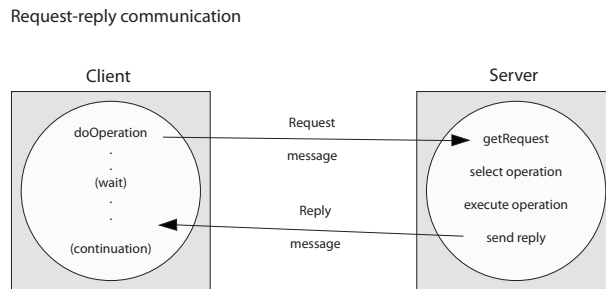
### 3.6.1 Request-reply protocols

This type of communication is widely used in client-server architectures. The client sends a request to perform an operation on the server, which executes it and sends the response back to the client. This type of operation can be *synchronous* or *asynchronous*, in the first case, the client stops executing and waits for the server's response, while in the second case, the client waits for the server's response, and continues with the execution of its process.

The figure 2.8 shows an example of a request-reply protocol, in which the client sends a request through the method `doOperation` containing the address of the server to send the request, the operation to be performed and any arguments required by the operation. At this point the client process waits for the response of the server or continues its execution, depending on the case that the protocol is synchronous or asynchronous.



Figure 3.8: Request-reply protocols



The server takes the request via the `getRequest` method. Once you get the request will analyze its contents, then select the required operation, executes it and sends the response back to the client.

Any messages exchanged through the request-reply protocol has an identifier that uniquely identifies it in the distributed system.

### Invocation Semantic

After sending the request `doOperation` by the client, if the server goes down or the request or response messages are lost, then there exists a time called *timeout* for which the client must wait for the expiration of this time, after which the client can begin various actions, depending on the *delivery guarantees* offered by the request-reply protocol in question. In the case of guarantees very low, after the expiration of the timeout, the `doOperation` does nothing and simply inform the client process that the request has failed. In the case of an higher delivery guarantee, the `doRequest` continues to send the request until it receives a reply. The number of attempts is finished, if after that number the client receives no response, the problem may be that which server has failed

In the case in which the request message is transmitted more than once

there is the risk that the server can perform the same operation requested by the client several times. For example, if the server receives a request message, but if the time to execute it and the time that the reply message reaches the client is longer than the timeout period, then the client will send a second request and the servant re-run the same operation. To avoid this type of problem, request messages have a request identifier in such a way as to be identified and to avoid that the server execute a duplicate request. When the server receives a duplicate request discards it and when it finishes executing the first request sends the response.

If the reply message is lost the protocols to have a greater guarantee of delivery allow sending from the client an acknowledge message, so the server receiving this message knows that the reply has arrived at its destination. If not arrive then returns the reply. The server then stores the reply to avoid having to start over.

So the request reply protocol can be classified according to their level of delivery guarantee in this way:

- the *request* (R) protocol. After the timeout the request ends.
- the *request-reply* (RR) protocol. After the timeout sends the request again.
- the *request-reply-acknowledge* (RRA) protocol. Use the acknowledge message.

### 3.6.2 Interfaces

The most modern programming languages allow to represent a program as a set of modules that communicate between them. The communication may be through procedure calls between modules or through direct access to variables in another module. However, if we wanted to impose the possible interactions between the modules then we can indicate which are the only procedures that can be called or just the only variables that can be accessed through an *interface*.

The interface hides the implementation of each procedure, and then in the course of time, the interface can remain the same even if the implementation of the procedures inside changes, without the user noticing.

It is not possible for a module in a process to directly access variables in a module of another process, at least that the interface permits the access.

- **Services interface:** In the client-server model, each server provides a set of procedures that are available for the use of clients. The services interface specification these procedures.
- **Remote interface:** In distributed object models a remote interface specifies methods of a remote object that is invoked by other processes. The other processes can then invoke only the methods that belong to that interface.

### 3.6.3 Remote Procedure Call (RPC)

Remote Procedure Call is a call pattern of procedures that allows a client program to call procedures in a server program that runs in separate processes generally on different computers from the customer's computer. The semantics of communication in the RPC are:

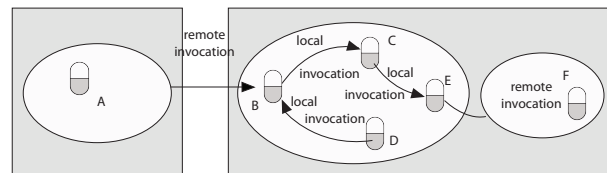
- **Maybe semantics:** with the maybe semantics the remote procedure may be called once or not at all. It does not apply any delivery guarantees. If the client does not receive a reply the request is not forwarded again, so do not know if the request has been executed once or the message of reply was lost, or not even that.
- **At-least-once semantics:** with this semantics if the invoker receives a response then know that the request has been executed at least once, otherwise an exception informs him that it have not received any results. These semantics can be reached via the retransmission of the message request. The server does not store the reply and there is no filtering for multiple request.
- **At-most-once semantics:** the invoker receives a result, in which case it knows that the method was executed exactly once, or an exception that informs him that no result has been received. These semantics is achieved by using all delivery guarantees.

### 3.6.4 Remote Method Invocation(RMI)

More recently the model in object-oriented programming has been extended to allow objects in different processes to communicate through the Remote Method Invocation (RMI). It is an extension of the local method invocations that allows an object in one process to invoke the methods of an object in another process. At this point we must distinguish between two types of invocation:

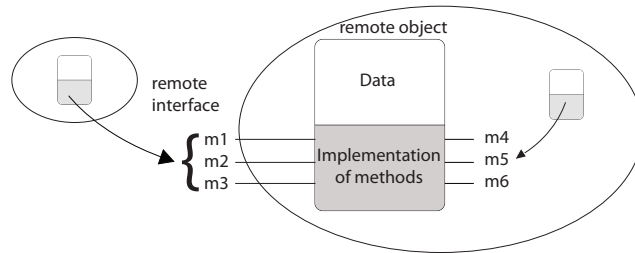
- **Remote method invocation:** it is an invocation to a method that belongs to an object of another process, whether it is or not on the same computer.
- **Local method invocation:** it is an invocation of a method of an object within the same process.

Figure 3.9: Remote Method Invocation



An object capable of receiving a remote invocation is called *remote object*. An object can invoke methods of another remote object only if it has its *remote object reference*. Using the remote object reference, you can specify a remote object uniquely in the distributed system. Each remote object has a *remote interface* with which specifies which methods are accessible remotely.

Figure 3.10: Remote Method Invocation



### 3.6.5 Events

The idea that underlies the use of events is that an object can react to change occurred into another object. For example, in an interactive application, the actions that the user performs in an object, such as pressing a button, are seen as events that cause changes in the state of the object.

Distributed systems based on events allowing multiple objects in different locations, to be notified of events that take place in a given object.

An object that generates events, *publishes* the type of events that it makes available to other objects. Objects that want to receive notifications from an object that has published its events, subscribe to the types of events that are of interest to them.

## 3.7 Processes' scheduling

The process scheduling algorithms are designed to optimize system performance while minimizing the average response time, maximizing the use of resources and balance the workload between the host. In dis-

tributed systems, **the processes queue is not unique**, so you have to decide who to allocate the process and if you need to migrate the process. There are different scheduling algorithms:

- **Centralized systems:** in this case exists a single processes queue and the scheduler is a component of the operating system with the task of managing the processes queue and apply a scheduling algorithm. For the centralized scheduling algorithms are: FIFO, round-robin and priority.
- **Probes algorithm:** when the processor is running on the process becomes overloaded, he decides to move the process. So probe eventually another processor, and if it is underused then it transfers the process in this processor.
- **Deterministic algorithm:** if we have  $n$  processes to distribute on  $k$  processors, with  $n > k$ , and we know the traffic between each pair of processes, then the algorithm distributes the load so as to minimize the total traffic between processes.
- **Centralized algorithm:** for each host is assigned a score. If a processor sends processes to other, the score increases, while if it get processes from other then the score decreases. A free processor receives processes from a host with a lower score.

## 3.8 Mutual Exclusion

When a collection of processes sharing a resource or a collection of resources, then mutual exclusion is required to prevent interference between the processes and ensure the consistency of shared data. In a distributed system this problem is called *distributed mutual exclusion*.

There are several algorithms to maintain mutual exclusion.

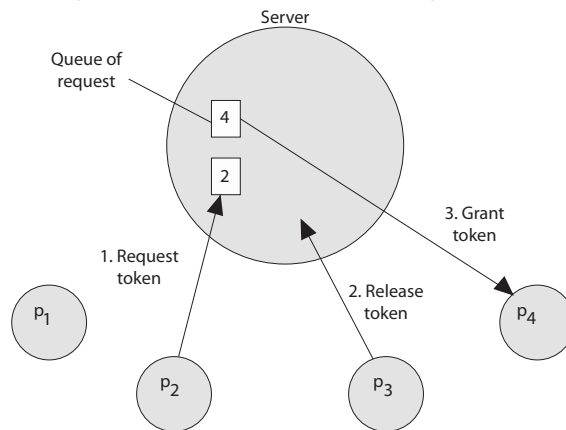
### 3.8.1 The central server algorithm

The easiest way to achieve mutual exclusion is to use a server that grants you permission to enter the critical section. To enter the critical section,

a client sends a request message to the server and waits for the answer. The response consists of a token that allows you to enter the critical section.

If the critical section is free, server gives you the token immediately, otherwise unresponsive to the client and queues the request. When the process that occupied the critical section free it, it releases the token to the server, which will pick the oldest request from the queue and sends the token to the client of that request.

Figure 3.11: Central Server algorithm



### 3.8.2 Distributed algorithm

A process that wants to enter the critical section sends a message to all the multicast with the following information: name of the critical section, its identifier, and local timestamp. Waits for the response of all. Once you get the OK from everyone, enters the critical section and as it leaves sends the okay to all the jobs in the queue.

A process that receives the request can:

- Not be in the requested critical section and does not want to enter in it, then sends the okay to the sender.

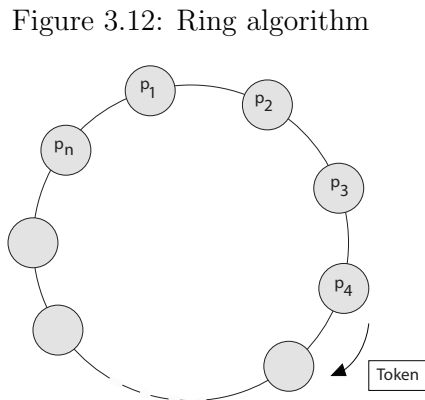
- Be in the critical section and then not responding. Puts the request message to a local queue.
- He also wants to enter in the critical section. It then compares its timestamp with that of the sender of the request. The oldest has higher priority. If the sender is the oldest sends ok, otherwise it places the message in the queue.

**Problems:**

- If there are N processes requires 2 (N-1) messages.
- If a process fails, no one else can enter in the critical section.
- Each process can be the bottleneck, because all the processes involved in every decision.

### 3.8.3 Ring-based algorithm

Order processes in circular order. At first, the process 1 has a token which then passes to the next. The process that has the token is enabled to enter in the critical section.





If  $N$  are the processes this technique uses  $[1, N-1]$  messages because a process obtains the token, 1 to get out and  $[1, N-1]$  to synchronize the critical section. If a process fails, you must reconfigure the ring, and if it fails, the process with the token you have to re-elect the next owner of the token.

### 3.9 Time

The time in distributed systems is an important and interesting problem to deal with.

The problem of time in distributed systems consists in being able to know the exact instant of time in which two or more events are successes on different machines, and consequently the possibility to understand the order in which they are successes or if they are successes simultaneously.

**The problem stems from the fact that in a distributed system, there is not a global time at which appeal.**

Each computer contains an internal physical clock, that is an electronic device that calculates the time within the computer. Unfortunately, the physical clock like everything else are imperfect. The difference between two clocks on different computers is called *skew*, while the difference of a given clock with an ideal clock is called *drift rate*.

*Coordinated Universal Time (UTC)* is a highly accurate external clock, based on atomic time, to which any clock can refer to synchronize.

The type of synchronization can be divided between:

- *External synchronization*: when the clock is synchronized with an external source, such as UTC.
- *Internal synchronization*: the clock between the various computers are synchronized between them with a certain degree of accuracy. Each computer understanding their level of accuracy compared to the other computers can refer to its own clock.

There are different synchronization algorithms.

### 3.9.1 Cristian algorithm

Cristian suggested the use of a time server, connected to a device that receives signals from a source of type UTC. This server provides the time at the request of clients. Upon receipt of the response, the client checks the clock time  $t$  just received. After that calculates the network delay  $t_{round}$  (the sending time of the request and the response) it sets its own clock as  $t + t_{round}/2$

#### Problem

If the central server falls is a problem, then you have multiple servers that the client contacts in multicast. Will use the first response it receives.

### 3.9.2 Berkeley algorithm

A computer coordinator is chosen to synchronize periodically the other computers. The computers send their clock value to the coordinator who value their local clock considering the round-trip time and calculates an average of the values obtained.

The coordinator instead of answering the current time date, which could introduce additional uncertainty due to the time of transmission, it sends the amount of time than the average just computed. The amount will be distinct for each computer. In short indicates who should speed up and who should slow down.

The value sent can be either positive or negative and the media does not consider the timing too far away or outliers.

### 3.9.3 Network Time Protocol (NTP)

The previous algorithms are designed for intranet, while NTP defines an architecture for a time service and a protocol to distribute information of time on the internet. Provides a service that enables clients through the internet, to be accurately synchronized to UTC, to provide a reliable service that can survive even with lengthy losses of connectivity: there are redundant servers and more roads between servers. Allows clients to

resynchronize frequently enough to compensate for the drift rate found in many computers. Therefore it is scalable, allowing a high number of clients.

NTP is provided by a network of servers allocated on the internet. Primary servers are directly connected to a source of time as UTC, while the secondary servers are synchronized with the primary server. The servers are connected in a hierarchical manner, and servers on the higher floors are more accurate because errors are introduced at every level.

### 3.10 Replication

For data replication means the preservation of copies of data on multiple computers. Replication is a key to make a more effective distributed system, as it is able to offer more performance, high availability and fault tolerance. Caching is different from replication, it does not necessarily increase availability.

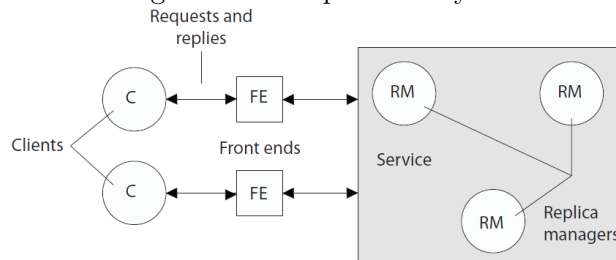
When the data are replicated, however, must comply with the transparency of replication: clients ignore the presence of multiple physical copies, they interact with a single serving. The replication also needs to be fault tolerant.

#### Model of a replication system

The model involves replicas held by distinct *Replication Managers*. They contain copies and run, ensuring the consistency of the replicas. The operations in a replication manager does not have to leave inconsistent result if these operations fail.

The set of replication manager may be *static* or *dynamic*. In a dynamic system, you may see new replication manager (for example a secretary copy a file into your laptop), instead this is not allowed in a static system. In a dynamic system the replication manager may fail, not a static one. Each client request is first handled by a component called *Front End*, its role is to communicate through message passing with one or more replication manager, instead of having it done at the client, it allows the

Figure 3.13: Replication system



properties of transparent replication. The requests are forwarded as well: the front-end issues request for one or more replica managers, ie either the front end communicates with a single replica manager who immediately communicates with the other replica manager, or the front end communicates in multicast with replication managers.

The replication manager will coordinate to decide the order of execution of a request in accordance with the other requests, all to ensure data consistency.

### Service fault tolerance

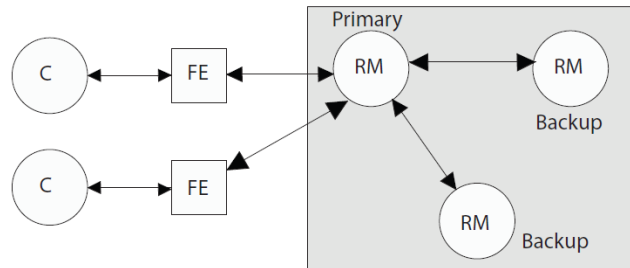
#### 1. Passive Replication (Primary Backup)

In the passive model or primary-backup for fault tolerance in the replication, there is a single primary replication manager and one or more secondary replication manager (backup or slaves).

The front end only communicate with the primary replication manager, which collects the requests in the same order in which they arrive. Check if it has already run once, if so then sends the response, otherwise executes the request and records the response. If it is an update request then sends a copy of the updated data to the backups, which should send an ack. The primary answer to the front end, which forwards it to the client.

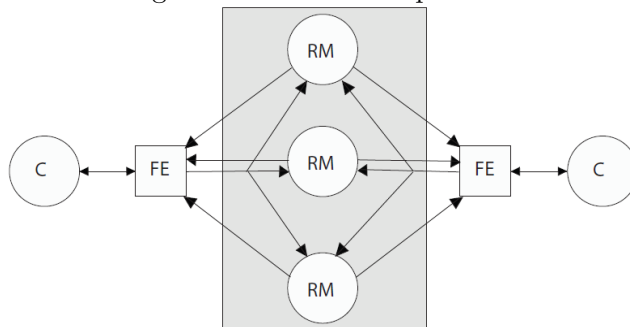
If the primary fails, one of the backups is promoted to primary.

Figure 3.14: Passive Replication



2. **Active Replication** In the model of active replication for fault tolerance, the Replication Manager all have the same role. The front end sends multicast their requests to all the replication manager, which process requests independently, but as the requests were sent to all in the same order, then it will be processed in the same way by everyone. Each replication manager sends the response back to the front end.

Figure 3.15: Active Replication



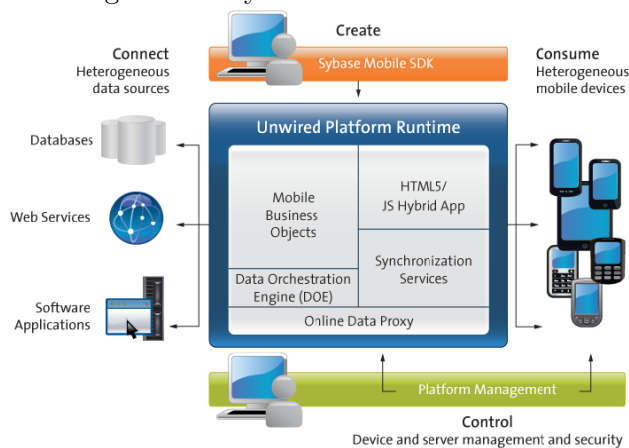
The requests of the front end are served in FIFO order (as the front-end waits for a response before making the next request) which is the same order of the program. This ensures sequential consistency.

# Chapter 4

## Sybase Unwired Platform

In this chapter we will discuss of a distributed system in particular, Sybase Unwired Platform (SUP). It allows a company having databases, Web servers or SAP servers that provide services within the scope of enterprise to expose them outside. So that a service offered by these servers can be accessed from any smartphone, tablet or any other mobile device.

Figure 4.1: Sybase Unwired Platform



Being a distributed system, faces many of the issues described in the pre-

vious chapter, for example scalability. In fact, allowing access to services to any mobile device, the numerous requests from outside could overload the server, so the SUP allows that does not happen, so that even a growing number of mobile devices can access these services.

This is just one of the problems facing the SUP, so we'll see how the issues set out in the previous chapter are resolved in a particular distributed system adopted by many companies worldwide.

## 4.1 Platform Solution

Sybase Unwired Platform acts as a hub that connects the back end of the enterprise system and its data sources to mobile device. It is essentially divided into two parts:

- **Sybase Mobile SDK:** the development platform that allows you to create your own application.
- **Sybase Unwired Platform Runtime:** the deployment and management architecture and services used to run and manage mobile applications.

This architecture enables the development and deployment of applications, allowing the connection between mobile devices of different platforms with different data resources belonging to heterogeneous back-end enterprise systems, wherever the latter is in.

So we can see that the SUP provides a middleware layer that can provide location, protocol, platform and programming languages transparency. Now we will see the architectural model of the platform SUP.

### 4.1.1 Sybase Unwired Platform Runtime

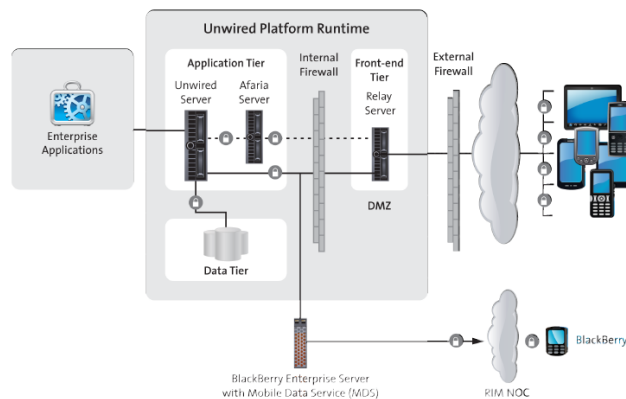
In this type of distributed system, there are one or more servers that provide shared resources and services. These servers are enclosed in an environment, called Enterprise Information System (EIS) or Enterprise Application.

Clients wishing to access the shared resources or services do not directly

access the EIS, but communicate with another server or cluster of servers, which grouped form the Sybase Unwired Platform Runtime (SUP). The SUP is able to provide the transparency necessary for a distributed system, such as transparency of competition, replication, fault, mobility, performance, and scalability. The SUP is also able to provide even the caching service.

Inside the SUP there is a server or a cluster of servers called Relay Server, which function as a proxy server for SUP and then in turn to the EIS. We can therefore deduce that the architectural model of the distributed system under consideration is proxy server, with or without multiple servers.

Figure 4.2: Sybase unwired Platform Runtime



The SUP is a distributed system, whose architectural pattern is on several layers, as defined in section 2.3.1. Regarding the tiered architecture, the three functional parts of an application are each assigned to a physical host, then we can say that it adopts the three-tier solution. Where the presentation logic is represented by the client, the application logic to the SUP and the data logic to the EIS.

In this chapter we focus on the lower layer of the vertical system, Computer and Network Hardware, while for the horizontal organization we focus on application and data tier.



## Unwired Server

Unwired Server belongs to the Application tier and manages the communication and data exchange process between the mobile devices and Enterprise Information System (EIS) data source. Optimizes access to the back-end system and it is interposed between the different mobile devices and the different data source of an EIS, it receives data from back-end systems and forwards them to mobile device, and vice versa. The Unwired server enables the connection between different platforms and operating systems, with different communication styles:

- **Replication-based synchronization:** a synchronization method in where cached data is downloaded to and uploaded from client to server via replication.
- **Messaging-based synchronization:** Data is maintained integrity on device client using a synchronization via messaging.

The unwired server can do this because it has a set of objects called Mobile Business Object (MBO) that can provide the transparency offered by the middleware layer. We'll talk more with precision in the next chapter.

## Data Tier

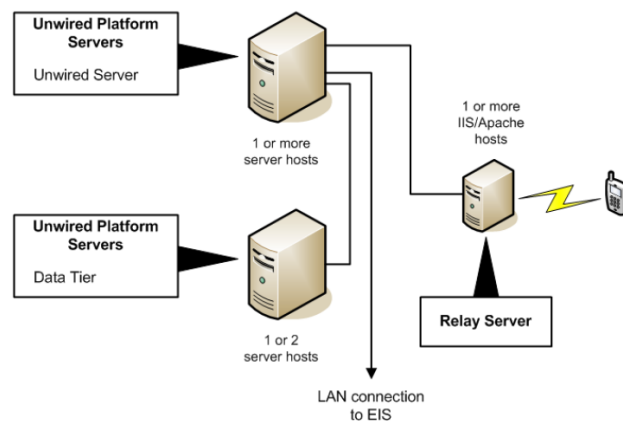
It belongs to the Data tier and it is a set of databases used by Unwired Server:

- **Cache Database(CDB):** retains the data retrieved from the back end. Mobile devices can access this cache to retrieve the data without having to log on several times a backend to always have the same data, allowing you to ease the workload on the back end. It offers the caching service provided by SUP.
- **Cluster database:** keeps all the information that allows the coordination within the cluster of Unwired Server.
- **Messaging Databases:** stores all asynchronous messages that then the application layer will have to process.

## Relay Server

The relay server behaves like a proxy server and allows secure communication and load balancing between Unwired Server and mobile devices. So it is able to ensure the scalability transparency. It allows the transit of data safely, without having to open any ports on the firewall, as well as to bypass any restrictions in the firewall.

Figure 4.3: Relay Server



It's a component of the runtime architecture within the enterprise demilitarized zone (DMZ).

- Provides a single point of contact between the mobile devices and the Unwired Server. Thus allowing the location transparency.
- Accepts requests from mobile devices and forwards to the Unwired Server.
- It is implemented as a pair of Web server extensions: one to manage communications with clients from the internet, and the other to manage the communications with the Unwired server in the internal LAN. Supports two types of Web Server: IIS on Windows and Apache on Linux.

- It is fully embedded within the security policies of Web servers, thereby avoiding the need to change the corporate policies of firewalls and IT.

The end-to-end communication between the mobile devices and the Relay Server using an encrypted communication via HTTPS. As said earlier, the relay server also provides a rudimentary load balancing in a platform Unwired Server cluster, distributing the requests of clients in a round-robin manner. So for process scheduling, relay server uses a classic algorithm of a centralized systems.

Each Unwired server has one or more **Outbound Enablers** (RSOEs) to manage communications with the relay server, while the relay server relies on open communication by RSOE to communicate with the Unwired server.

The RSOE is an Unwired server process in a particular Unwired server port, it opens outbound HTTPS connections to relay server, for communication both incoming and outgoing with Unwired server.

You can also use any hardware or software **load balancer** to improve performance, along with the relay server.

## 4.2 Cluster and Non-cluster systems

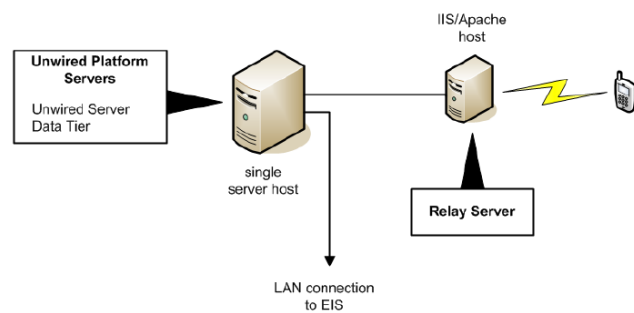
The SUP allows to obtain, in case of need, a model with multiple servers. Thus ensuring scalability, performance and fault transparency.

So we can install a single server, or more than one, so we have to distinguish between Clustered and Non-Clustered Systems.

- **Non-clustered Systems:** all Unwired Platform server components are installed in a single host. A non-clustered Unwired Platform Runtime system is simpler and less expensive to maintain, however, has many limitations:
  - Is not scalable, so you can not add or subtract servers to adapt the system to changes in load and performance requirements.
  - Is not able to provide a load balancing and a mechanism for fault tolerance.

- The only way to increase the performance of the system is to increase the potential host: CPU, RAM, etc.

Figure 4.4: Non-clustered systems

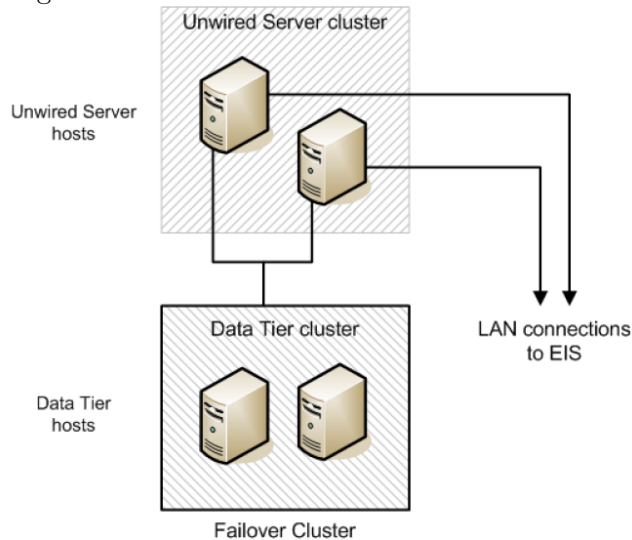


- **Clustered Systems:** All the benefits that a non-cluster does not provide, are offered by a cluster of server: scalability, by adding or subtracting the server, to adapt the system to changes in load and performance. Multiple servers also allow for load balancing and fault tolerance mechanisms. So if we want a scalable system with higher availability and higher system performance we have to choose a system of cluster type.

## 4.2.1 Cluster Types

In an Unwired Platform system there are two types of clusters: Unwired Server cluster and Data Tier cluster, and Unwired Platform system is a cluster if it include at least one of the two type of cluster.

Figure 4.5: Unwired Platform servers in clusters



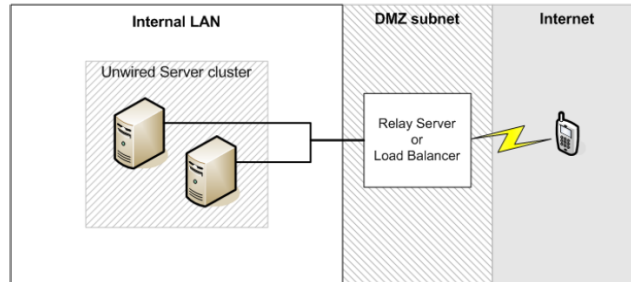
### Unwired Server cluster

The Unwired Server cluster enables load balancing of client request, sharing the client workload, moreover it improves the system availability and its performance.

Each server in the cluster serves the same set of clients, users, and mobile devices and shares common data tier resources and rely on the same set of EIS, so all Unwired Servers in the cluster have access to the same cached data from the EIS, messaging data for clients and cluster and server configuration data.

This data sharing enables the Unwired Server cluster to scale easily by adding or removing servers at any time.

Figure 4.6: Unwired Server cluster



In the Unwired Server cluster the application is deployed to multiple servers within the cluster, but only one copy at a time is online. As that application fails, or that server goes down, the application is restarted on another server of cluster. This is a form of *replication* of type **Passive Replication**.

Moreover, thanks to the Relay Server or load balancer the Unwired Server can share the workload to improve performance and efficiency, and give the client a single point of access, regardless of the server in the cluster that will access.

### Data Tier Cluster

Data Tier Cluster provides support in case of failure to improve system availability and fault tolerance.

The Data Tier Cluster consists of at least two hosts, one is redundant: one is active and other passive. The cluster has a shared directory that all hosts have permission to read and write.

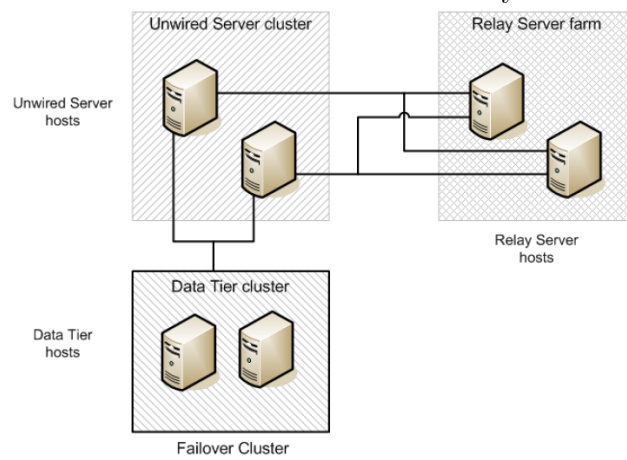
So we can see that the Sybase Unwired Platform (SUP) uses *data replication* with a *fault tolerance service* of type **Passive Replication**.

### Relay Server Cluster

To further improve load balancing and system availability, it is possible to use a Relay Server cluster. It consists of two or more relay server, each

of which serves the same set of  $RSOE_s$ , which in turn is associated with a specific Unwired Server.

Figure 4.7: Unwired Server connection to Relay Servers in cluster



### 4.3 Mobile Business Object (MBO)

As we have said many times the Sybase Unwired Platform must ensure communication between devices with different platforms, such as Android, IOS, Windows Phone, BlackBerry, and different data sources like SAP server, Web Server and different type of databases. This communication is guaranteed by Mobile Business Object (MBO), that in the vertical organization of services it represent the **middleware layer**. So MBO is able to ensure communication offering Platform and Programming Languages transparency. In addition to the location transparency, since the mobile device does not know where is the Enterprise Information System (EIS), but they connect to EIS through the MBO in the Sybase Unwired Platform(SUP).

The Mobile Business Object is the cornerstone of all the architectural solution of the Sybase Unwired Platform.

Represents the *business logic* in the three-tier solution, stored in the Unwired Server, defining the data that must be taken from the back-end system to be displayed to the user through the mobile application.

MBO represents an object data models that defines connections to the back-end EIS systems, operations, attributes, and relationships that allow application mobile to filter from the back-end system only data that it want and synchronize data with mobile devices. Infact, MBO is considered an **interface** that indicates which of the possible data or operations provided by the EIS can be used from the mobile application. MBO indicates the input arguments to be delivered to the remote procedure and what should be the output data, but the implementation of the procedure remains completely unknown. So in the time, implementation could change, for example, could be updated and made more efficient, without the application, which connects to the MBO, should be modified. The important thing is that you do not change the interface of the procedure whereby MBO is connected to it.

As stated in Section 2.5.2, there are two types of interfaces, services and remote interface, MBO is a **services interface** as it is a client-server model, where clients demand the execution of a procedure in the EIS.



The MBOs are deployed in the Unwired Servers and they are accessed by mobile applications from client devices.

The main task of the Mobile Business Object is to provide a layer of abstraction to allow the iterations between different types of mobile devices with different types of back-end data sources, as depicted in the figure:

Figure 4.8: Mobile Business Object



They are reusable, as if a new mobile device is added, it can use the same MBO, allowing the system to be scalable.

Once implemented the MBO, are placed in a package. This package is deployed inside the Unwired Server.

Now, how do mobile applications communicate with the MBO within the Unwired Server?

Each mobile application is developed using special libraries, called Object API, developed for each mobile platform that can run the application: Android, iOS, Windows Mobile and Blackberry. Such Object APIs allow the mobile application to communicate with the MBO, providing data transport services, reliability and security, exchange of messages and notifications in case of change of data occurred in the back-end or in the mobile application. In short, all those services to be able to synchronize mobile devices with the data inside the Unwired Server, and to keep the data in the EIS, Unwired Server and mobile devices in a consistent state,

as we will see later. *So the Object API is an interface for all mobile platforms, to communicate with the MBO in the Unwired Server.*

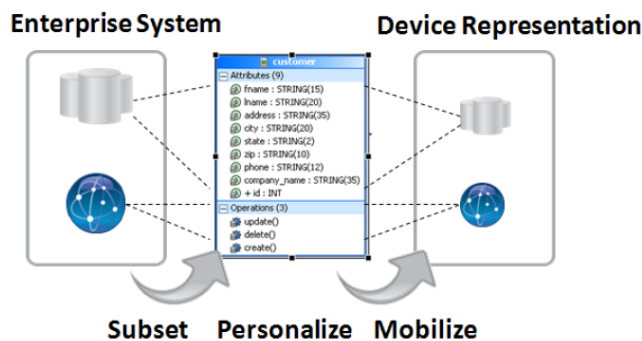
Now we can understand as the Sybase Unwired Platform is a three-tier solution, as the three functional parts of an application are displayed on three different hosts: the presentation logic on mobile devices, the application logic, represented by the MBO is deployed in the Unwired Server, while the data logic is in the EIS system.

An MBO also offers the service of virtualization, that normalizes the data to allow the interaction between different enterprise information systems (EIS), each having its own interface connection, data, operations and data structures.

### MBO Attributes

The data inside the EIS are represented by tables. A Mobile Business Object (MBO) is associated with only one of these tables. The columns of this table are the attributes of the MBO, so the MBO attributes define the structure in the MBO of the data associated with the EIS and mobile application. Attributes define the scope of the device-side data store.

Figure 4.9: Mobilize EIS data



## Operations

MBO can incorporate operations that can or can not change the retrieved data from enterprise information system (EIS).

They are of various kind:

- **Create, Update, Delete(CUD)operations:** the arguments of these operations are mapped to the arguments of the operations of the EIS, and can create, update and delete data, and cause the change of state of the MBO.
- **Read/Load:** operations that load data from EIS to the cache of Unwired Server.
- **OTHER:** any other type of operation that do not cause state change.

MBO contains the operations that the client may request to the EIS. The client requesting the execution of an operation, sends a request through the **request-reply protocol** to the server that contains the MBO in question. After that, the request is forwarded to the EIS, again via the request-reply protocol. Then the client does nothing else to make a **Request Procedure Call (RPC)** to the EIS through MBO. Among the three invocation semantics, set out in section 2.6.3, SUP uses the one with more guarantees, ie at **most once semantic**.

Analyzing how the request is made from the point of view of the vertical and horizontal system, described in sections 2.4.1 and 2.4.2, we can see how the request part from the *application layer* in the tier relative to the *presentation logic*. The request goes down to *middleware layer*, in which the request is reformulated through Object API and reported at the *lower-layer network*. Arrived at the SUP, that is the tier relative the *application logic*, from the network layer, go to middleware, ie the MBO, which in turn forwards it to the EIS, that is the tier relative the *data logic*, passing through the network layer. When it reaches the EIS ascend the layers and finally executed. The answer will retrace the same route in reverse.

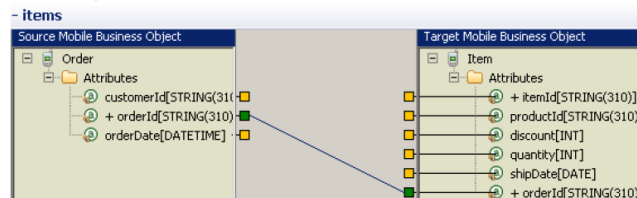
## Multiple Mobile Business Object

A Mobile Business Object (MBO) is associated with a table in the EIS, but you can create multiple Mobile Business Object (MBO) from a single read operation, which allows selection of *multiple output tables* from a single call to the Enterprise Information System (EIS) to which the MBO is bound. To create a multiple mobile business object we need a read operation that returns more than one table in their output. Each table is an independent result set and will be mapped to a corresponding MBO. The MBOs created from a single read operation are treated as a single block, so as to be able to upgrade all MBO performing the operations EIS once.

## Relationship

A relationship define the data association between two MBO by linking attributes and load arguments (the read operation's parameters, we will see in detail in the next section) in one MBO to attributes and load arguments in another MBO.

Figure 4.10: Bind attributes of two MBO



Relationships help to provide the data as one unit, and properly sequence the operations on the related MBO.

Relationship type allowed are:

- One to many



- One to one



- Many to one

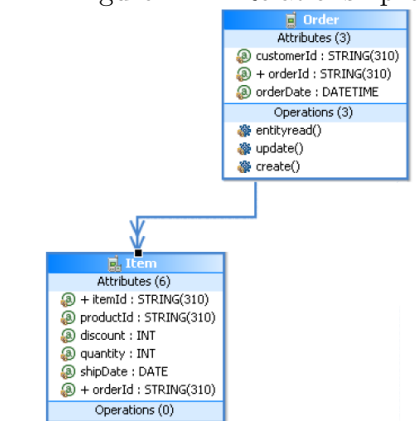


- Bi-directional



- **Composite:** changes in the parent MBO are propagated to the child's MBO.

Figure 4.11: Relationship between MBO



## Personalization Key

Personalization Key allow the mobile user do define certain input field values within the mobile application, by associating his name with a

simple or complex datatype value.

### **Object Query**

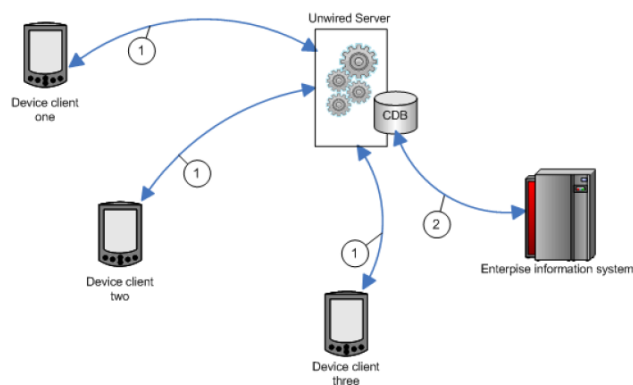
Object queries are SQL statements associated with a mobile business object (MBO) running in the mobile device. They return a subset of the data of the MBO. For example, an object query is used to filter data already downloaded from the CDB to display a single row of a table when triggered.

## 4.4 Data Synchronization and Data refresh

Inside the Enterprise Information System (EIS) changes may occur in the data, such as mobile devices could change the data within the Mobile Business Object (MBO) in the Unwired Server. So there is a need to maintain data consistency between the EIS, SUP and mobile devices. Therefore SUP provides techniques to update the data and make consistent the data held by mobile devices with the copy of the data stored in the Unwired Server Cache(CDB), before copying everything in the Enterprise Information System (EIS). These techniques are called:

- **Synchronization:** synchronizes the data in the Unwired Server Cache with the data in the mobile devices. It is essential to keep data synchronized between Unwired server and device client.
- **Refresh:** synchronizes the data in the Unwired Server Cache with the data in the Enterprise Information System. If the connection or the enterprise information system fail, however, the devices would still have access to the data in the Unwired Server cache.

Figure 4.12: Data Synchronization and Data Refresh



1. Each mobile device requests data from the Unwired Server and keeps one copy of the data. Even Unwired Server Cache and Enterprise Information System maintain a single version of the data.

2. If changes occur in the enterprise information system, through the refresh also Unwired Server Cache is updated, after which, through synchronization, mobile devices will be updated.

Along the section we discuss in more detail of both techniques, but first we need to better understand what is the Unwired Server cache.

#### 4.4.1 Unwired Server Cache

The Unwired Server Cache (or cache databases CDB) caches data required by device application.

The data in a cache can have two states: valid or invalid. In the second case, are no longer relevant and can be overwritten. The advantage of an hardware cache is that is extremely fast, as the data can be inserted or deleted inside extremely rapidly. So when the processor at any given time who need them, can get them quickly, without having to go get them in main memory.

The CDB however, is different from a normal cache hardware, as while the second is always consistent and is even able to replace the system memory, the CDB is not able to ensure the consistency with the EIS, for example, if the connection between EIS and SUP goes down, clients continue to communicate with the SUP, but it is not said that the data into it are still consistent with the EIS.

The data residing in the CDB is used only to detect data changes between SUP and mobile device, so only the difference is transferred to the device.

The CBD, however, performs some important functions:

- Contains a local copy of enterprise data recently used.
- Manages updates between CDB and EIS server (data refresh).
- Manages updates between CDB and mobile devices (synchronization), even in the case of thousands of simultaneous synchronizations.



- Is able to create partitions of the data belonging to a MBO, for example, the user is able to provide a particular value of a parameter mapped with a given attribute of the MBO, so as to filter and return to the client only the data corresponding to that value. You will also create a partition in the CDB containing these values, for future requests. In addition, multiple partitions allow simultaneous synchronizations, as we will see shortly.

Managing the Unwired Server cache ensures that enterprise data remains stable, available and consistent across sources.

Thus we can see that many mobile devices, through the SUP, ask to read and modify the data in the EIS. So what the SUP have to do is to ensure **mutual exclusion**. It guarantees the mutual exclusion in a centralized manner, as described in Section 2.8.1, through the CBD, whose task is to maintain the consistency of data.

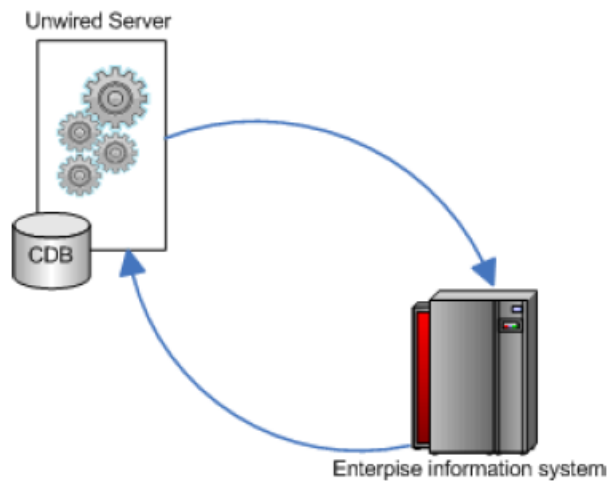
A cluster system can have any number of Unwired Server, but only a CDB shared among all. More number of Unwired Server increases and the more the number of worker threads dedicated to CDB increases.

#### 4.4.2 Data Refresh

Data Refresh is the technique which maintains data consistency between the Enterprise Information System (EIS) and Unwired Server cache (CDB). A Data Refresh occurs when EIS data updates are propagated to the CDB. There are two types of Data Refresh:

- **Cache Refresh schedules:** Update data in the unwired server cache with the latest data from the Enterprise Information System at scheduled intervals or on demand.
- **Data change notification (DCN):** when the data used by the application change, then the Enterprise Information System notify the Unwired Server that it needs to synchronize itself with the updates, and mobile devices with such modifications.

Figure 4.13: Data Refresh



When a refresh is called, all the rows returned by the Enterprise Information System are compared with the rows that already exist in the CDB in the following manner:

- If the CDB is empty, it is filled with all the rows returned by the EIS.
- The Unwired Server scans the rows returned by the EIS through the primary key, and check if these rows already exist in the CDB
  - Refresh the rows that are different and the next synchronization called by mobile devices, returns only the rows updated, rather than return them all. This is to increase performance and efficiency.
  - Rows that do not exist are inserted

Data Refresh behaviour for every mobile business object (MBO) is specified in a group, called *Cache Group*, in which reside the MBO that share the same behavior. This behavior is specified by a *Cache Policy*.

## Cache Policy

Specifies how the Mobile Business Object (MBO) within a cache group must be loaded and updated from the Enterprise Information System (EIS) to the Unwired Server cache (CDB).

Setting a cache policy for an MBO allows you to control the interactions between the EIS and Unwired Server cache. To better manage these interactions improves application performance.

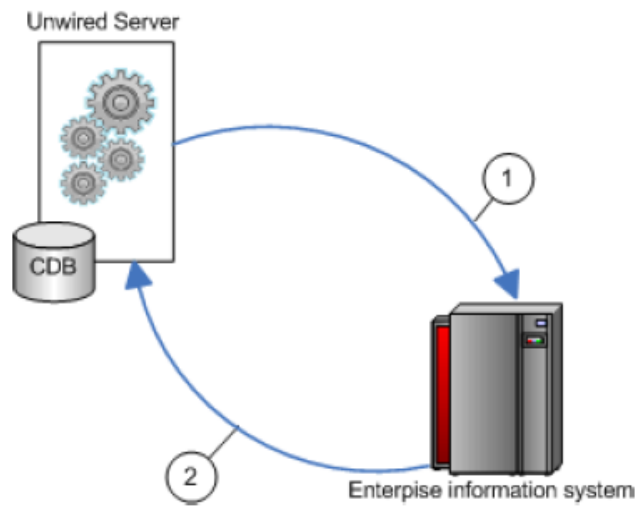
### 4.4.3 Cache Refresh schedules

In this case the refresh requests are sent through the *request-reply protocol* with an invocation of type *Request Procedure Call (RPC)*, and we can have two types of policy:

- **On-Demand:** the cache becomes invalid after a certain period of time, called *cache interval*. The cache after this interval is not immediately updated, but expect it to be a request made by the user. In order not to overload EIS, if not necessary. The refresh request is done through an invocation of type Request Procedure Call (RPC), as it is requested by the client a read operation to the EIS through MBO.
- **Scheduled:** the cache becomes invalid at the end of the cache interval, but then this time the Unwired Server immediately asks the updates to the EIS. This technique is used when the cache interval is large and the data during this period do not have great need to be updated. So, is not required consistency between the data during the cache interval.

Configure Unwired Server to “poll” the enterprise information system (EIS) at scheduled intervals to determine if data has changed. If it has, the EIS refreshes cached MBO data.

Figure 4.14: Data Refresh Initiated by Unwired Server



1. Unwired Server polls the EIS at an interval determined by the Unwired Server administrator.
2. If data changes, the CDB is refreshed.

This simple and flexible data refresh strategy uses more system resources than data change notification (DCN).

With the On-Demand and Schedule policies, all MBO within the same cache group are updated in block. So more data contains the MBO and the longer it will take to update the cache. If we did not want to update all the MBO within a cache group, then we should divide the cache group favors the needs of updating of various MBO. This makes update operations less heavy and therefore improves the performance.

The division of MBO between multiple cache groups, allows the parallel upgrade between more cache group, therefore decreasing the update time than updating a single cache group with all the MBO in.

Thus the division of MBO in more cache group provides the *performance transparency*, since, if the number of MBO at some point begins to be too high, just divide them between multiple cache group. This allows you to update in parallel multiple MBO. All this, as already mentioned, increases the performance without having to radically change the system.

### **Cache Interval**

The cache interval is used both in Schedule and On-Demand policies, and can have various widths: hours, minutes, seconds, and so on.

If a user performs a synchronization before the interval cache expires, then it will receive the data from the CDB, *without making a request to the EIS*, as the data in the CDB are considered still valid.

If you make a synchronization after the interval cache has expired, then the data within the CBD are considered invalid, then the CBD have to synchronize with EIS. Such request shall be made at the end of the cache interval with the schedule policy, when the request is made with On-Demand policy.

With On-Demand policy you can set the cache interval to 0, so when the user performs the synchronization request, the CDB requests data from the EIS and then forwards them to the user. The data within the CBD are immediately rendered invalid, so at the next synchronization request, the data is requested again to EIS.

This technique ensures the user to receive the updated information with EIS, ensuring a maximum consistency of the data, but if the requests are many, then it increases the workload for the EIS and the response time, so we have a reduction of the performance.

Moreover, with this technique the response time increases, because every time the request of Synchronization is forwarded to the EIS.

### **Load Arguments and Cache Partitions**

A Load Argument can be considered as an argument passed to the refresh request. It allows you to download from the EIS only the data indicated by the value assigned to him.

Each Argument Load creates a partition in the CBD, based on the value

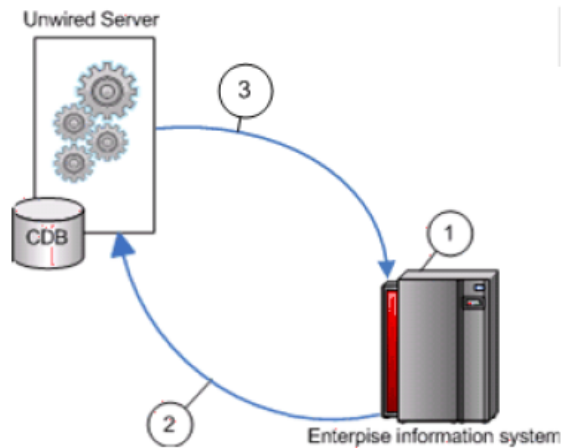
assigned to him. By creating multiple partitions within the CDB, allows a refresh parallel and independent of the various partitions, thereby increasing the refresh rate. For example, you can refresh multiple partitions in parallel, or query one partition while another is being refreshed. In general, partitions prevent serialized access to the cache. So we have a considerable increase in performance.

Partition granularity is an important consideration during model development. Coarse-grained partitions incur long refresh times, whereas fine grained partitions create overhead due to frequent interaction between EIS and Unwired Server.

#### 4.4.4 Data Change Notification (DCN)

In this case the refresh requests are not sent via an invocation of type Request Procedure Call (RPC), as in the Data Refresh schedule, but by events. In fact, when there will be a change in the data of the EIS, **an event will be triggered**, and notified the SUP.

Figure 4.15: Data Change Notification



1. A change of data occurs in the EIS.

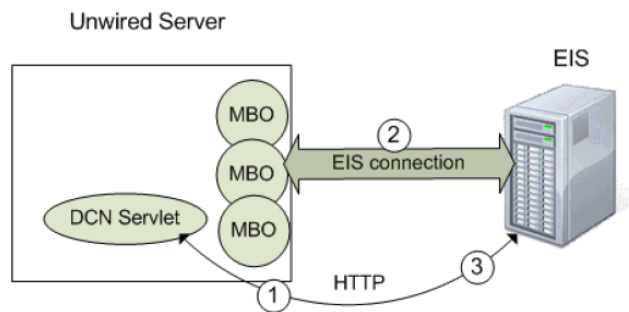
2. The DCN notification is issued to Unwired Server.
3. A result response is returned to the EIS.

Date Change Notification (DCN) is an update mechanism that allows an EIS to send updates to the CBD, as soon as they happen. So it is no longer Unwired Server cache (CDB) to ask for an upgrade to the EIS, but the EIS that sends updates to the CBD. The data in the CDB never become invalid, because with the DCN as soon as there is a change in the data of the EIS, they are immediately updated.

### DCN with or without Payload

In the DCN without payload, in the DCN request is included the operation of the MBO, which Unwired Server have to run to update the date.

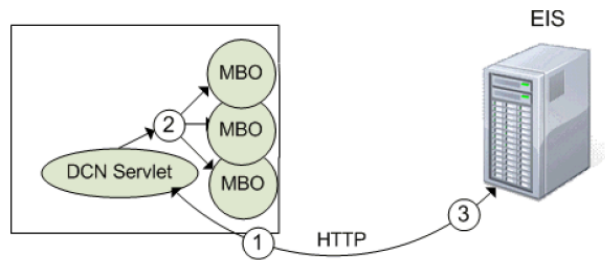
Figure 4.16: Data Change Notification without payload



1. The DCN sends a request to perform an operation of an MBO, including the parameters.
2. Unwired Server performs the operation and updates the data.
3. Unwired Server sends an ack to the EIS.

While in the case of DCN with payload, EIS sends the updated data directly, and Unwired Server only needs to perform the insert, update, or deletion of data, to perform the upgrade.

Figure 4.17: Data Change Notification with payload  
Unwired Server



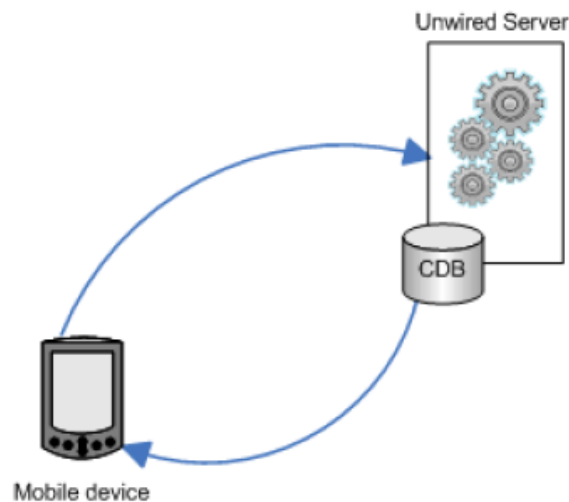
1. The DCN sends the new data.
2. Unwired Server insert, update or delete data.
3. Unwired Server sends an ack to the EIS.



### 4.4.5 Synchronization

The synchronization allows the mobile device to filter the data to upload and download from the Unwired Server cache (CDB) and to decide how often to do so. Is done by invocations of type *Remote Procedure Call (RPC)* using the request-reply protocol. It allows to maintain consistency between the data held by mobile devices, and those contained in the Unwired Server cache (CDB).

Figure 4.18: Synchronization

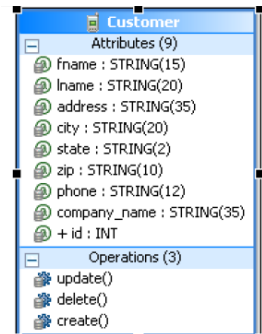


#### Synchronization parameter

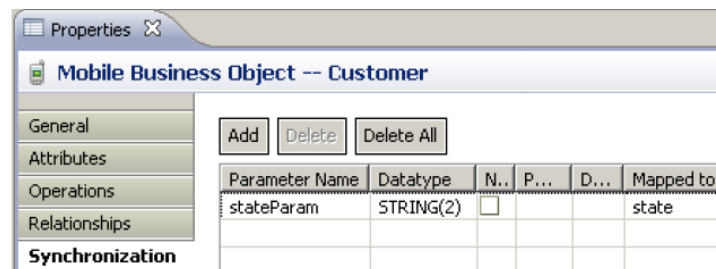
The mean that allows you to filter data is the Synchronization parameter by which you can reduce the number of rows downloaded from CDB to the mobile device.

When an MBO is bound to a data source, the attributes of the Mobile Business Object (MBO) are mapped to the database columns of the data source. Then you can control the amount of data to be filtered by defining a Synchronization parameter mapped to the attributes of MBO.

Here's an example with an MBO, called Customer:



of which only attribute “state” you are interested to synchronize:



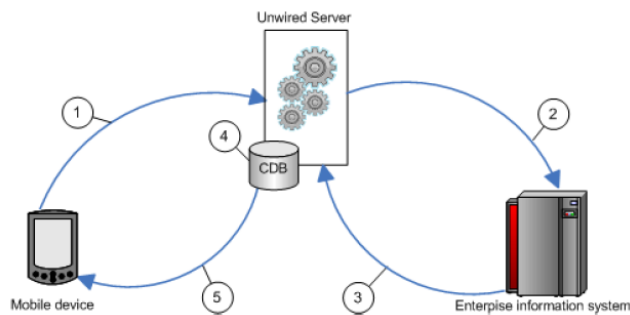
Filter the data to be downloaded to the mobile device can be useful for those MBOs which possess a large amount of data that does not change frequently. Then they are downloaded wholesale in the CBD or updated with a cache interval very wide and filtered from the mobile device through a Synchronization parameter to extract only the data of interest. If you do not define synchronization parameters, a client may download all data in the CBD, and, in the worst case, cause device application to crash.

Synchronization is also possible to map a parameter with a Load Arguments, so as to partition the CBD through a synchronization request. If the data are valid in the CBD, then there will not be a refresh.

Pairing a load argument with a synchronization parameter indicates that the user will supply values for this argument over time and the aggregate

set of data based on the values provided over time are synchronized with the mobile device.

Figure 4.19: Combine Synchronization and Data Refresh strategies



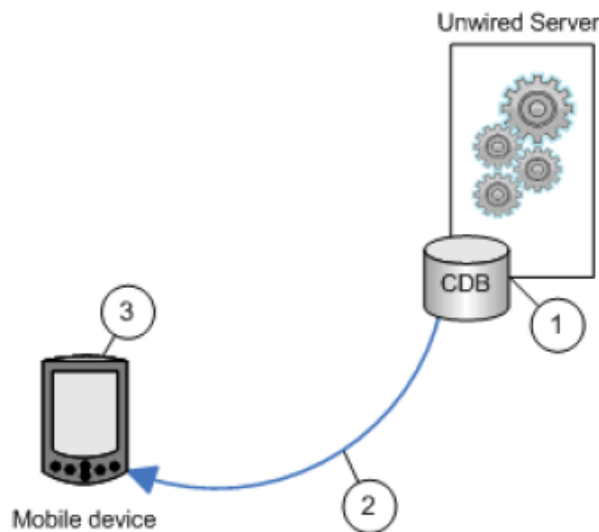
1. The user starts a synchronization providing a value to the Synchronization parameter, for example through a Personalization Key.
2. Load Argument is mapped with the same Personalization Key, and Unwired Server passes the query to the EIS.
3. The EIS sends to Unwired Server the updates.
4. Unwired Server creates a partition with the result in the CDB or updates the partition if the user has previously synchronized.
5. Unwired Server synchronizes the device with the data in the CDB partition for the user.

This provides more fine-grained CDB partitioning and concurrency, but may introduce more partition refresh overhead.

### **Synchronization initiated by Unwired Server**

Synchronization can also start from the Unwired Server instead of the mobile device.

Figure 4.20: Synchronization initiated by Unwired Server



1. Unwired Server becomes aware of a change in the data in the CDB, for example by means of a data change notification or a data refresh.
2. Unwired Server notifies the mobile device that has been a change in CDB. If configured, the server may force synchronization.
3. If the server does not force synchronization, then the application will be implemented to synchronize when it receives a synchronization notification.

### Synchronization group

A synchronization group specifies the synchronization behavior for all MBO within it. If the mobile device initiates a synchronization to an MBO within the group, then all the others MBO<sub>s</sub> within the group are synchronized.

Through a synchronization group you can control which MBO synchronize, thus allowing also a prioritization of the synchronization, ie to decide which MBO synchronize before other. It thus provides a form of *mutual exclusion*, as applied to different mobile applications an order of access to shared resources.

Within a synchronization session you can specify multiple Synchronization group. A session with more than one group is more efficient than more sessions with a group only. More sessions means more transactions and more overhead.

## Chapter 5

# Implementation of an HTML5/JS Hybrid Application

In this chapter we discuss of the highest layer, the *application layer*, in particular describe how you create a hybrid application using the Sybase Unwired Platform (SUP). The main types of applications that can be built using SUP, are native applications and HTML5/JS Hybrid Application. A native application is developed with a code for the mobile platform on which it will run, such as Objective-C for iOS, Java for Android. By using the Object API, which we discussed in section 3.3, the native application is able to communicate with the SUP and the Mobile Business Object (MBO) inside.

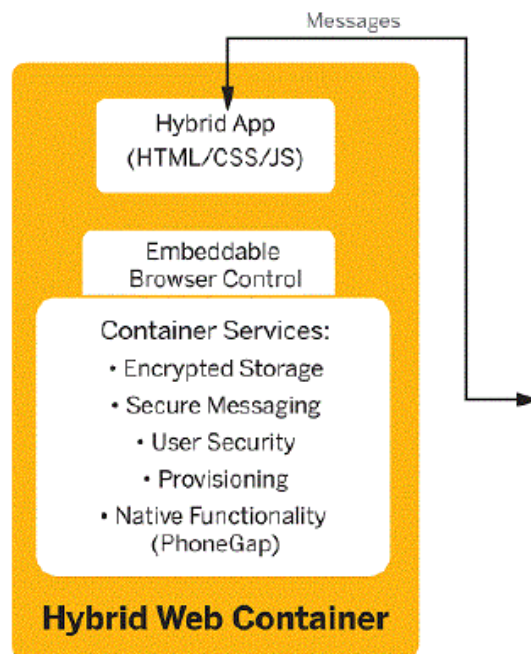
This type of application then uses compiled code for a particular operating system, depending on the platform on which it must run, and its implementation is extremely flexible, because it uses native code. It also provide offline capability. The problem is that a native application must be provided to each individual mobile device after being completed, even small changes must be reported individually on each mobile device.

An HTML5/JS hybrid application is instead a fully generated Web application package. So it is an application developed in HTML5, JavaScript and CSS, completely independent of the platform on which it should run.

After that this package is automatically deployed within a container in the mobile device. This container is built with native code of the platform on which it will run.

The container is an embedded browser that contains the Object API with which it can communicate with the MBO<sub>s</sub> in the SUP and provides all the services necessary for the application, such as connectivity, guaranteed and reliable messaging, caching and security.

Figure 5.1: Hybrid Web Container



Data transport and access relies on messaging protocol of type *request-reply protocol* between the SUP and the container on the mobile device.

An HTML5/JS hybrid application allows the development of simple applications, compared to the native ones, but utilize the power of native

device services, because by using the HybridWeb Container for each type of platform in a mobile device you can create a single HTML5 application that performs advanced device specific operations on all the different devices.

Another advantage of a hybrid application is that it allows you to write code once for all platforms. With this approach, a hybrid web container is developed and deployed to a device, then one or more hybrid applications are deployed to the container.

So in essence, the container can be seen as a service interface with which the hybrid application can communicate with the MBO in the Unwired Server. *It corresponds to the middleware layer in the stack of layers of the mobile device.*

## **5.1 The purpose of the application.**

The hybrid application that we are going to develop, connects our mobile to the CRM servers of SAP in the CRM test environment of the Bocconi University. The application will be able to request to the server the Business Partners (BP) of Bocconi, person and company, that the user wants by getting their personal data. Furthermore, given a BP, we could request the leads associated with it, and change their state. In the end we could create an activity of a given BP.

To lead means an event that happened at the BP, as a donation received to BP, while activity means a real activity that BP has made or plans to make such a course to play at the university.

## **5.2 Implementation of Mobile Business Object**

Firstly you must implement the Mobile Business Object, which must be deployed in the Unwired Server. As mentioned in the previous chapter the Mobile Business Object (MBO) is in the stack of layers of the archi-



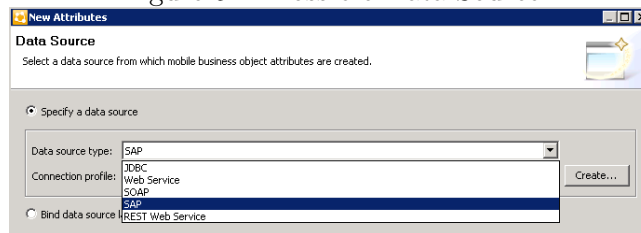
tectural pattern the **middleware layer**, providing the transparency of location, platform, protocol and programming language. In this section we show how it is implemented a simple Mobile Business Object, trying to highlight the transparency offered by it.

Among the three functional parts of an application the MBO is part of the **application logic**, which will be deployed in the Unwired Server in the **application tier**.

### 5.2.1 Data Source and Profile Connection

MBO needs to know which data source in the Enterprise Information System (EIS) must be bound and hence such data, in the form of table of attributes, must represent. Those possible are shown in figure.

Figure 5.2: Possible Data Source

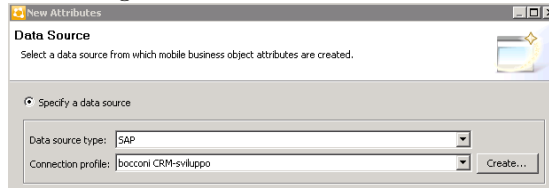


Once loaded into the Unwired Server, the MBO will need to know the location of the EIS, in order to connect to it. The location information will be provided through a Connection Profile. So we will create a Connection Profile, informations of which we do not show for reasons of privacy, where we will indicate the location of the EIS. The Connection Profile will call Bocconi CRM-development.

Here we once again highlight the **location transparency**, in fact, mobile devices will not know where is the EIS which require data and services, but rely on Mobile Business Object. So the location of the EIS can change, without that mobile devices noticing, just need to modify the Connection Profile of MBO in the SUP.

In addition to the location transparency, here it also becomes more evident that the MBO offers the **platform and programming language**

Figure 5.3: Connection Profile

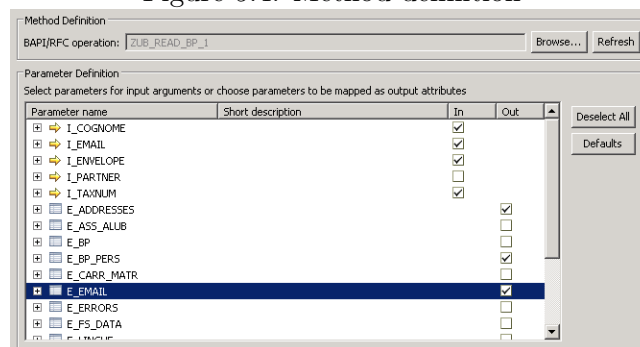


**transparency.** Indeed, the data source offered by EIS are more than one. This, however, is completely indifferent for mobile devices, which by their connection to the data source does not have to make any kind of discrimination.

### 5.2.2 Method Definition

Suppose we want to build the MBO relating to person BP. The function in the EIS that allows the reading of the person BP is called ZUB\_READ\_BP\_1. At this function, you must provide the input parameters and it will return values in output. The values in the output are tables containing the personal data of BP. These data are sorted by attributes, such as name, surname, etc..

Figure 5.4: Method definition



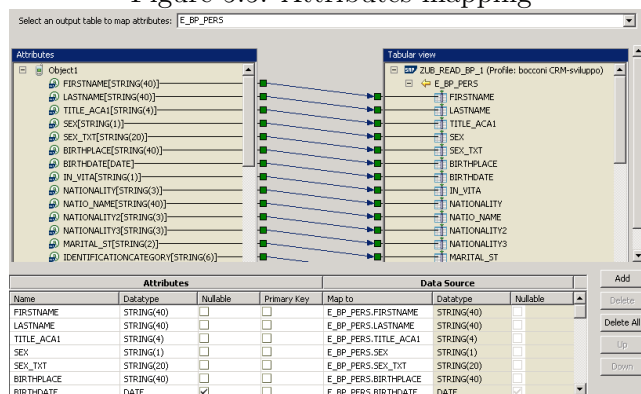
From the figure you can see that you can choose the input parameters

and output data desired.

So the mobile device will make the **remote invocations** to the EIS through the MBO and the function that will call for BP to have the desired person will ZUB\_READ\_BP\_1. What is shown in the figure is the *interface* of this function, the device or the mobile application know nothing about how it is implemented, they can only request it, giving any input parameters and requests the tables you want to get out. The calls of type **Remote Procedure Call** will refer to such **services interface**.

From the figure below:

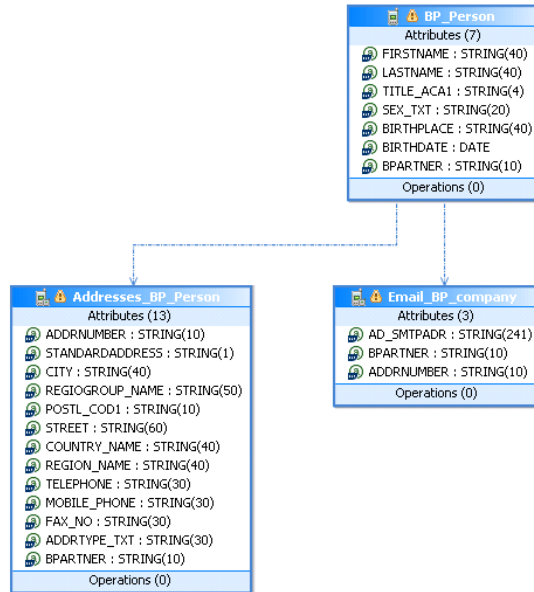
Figure 5.5: Attributes mapping



we can see on the left the attributes that form the MBO, while on the right we have the table attributes E\_BP\_PERS output returned by the function. These links are indifferent on the type of EIS data source chosen, thanks to the **transparency of the platform**. Whatever the chosen EIS data source, the data returned by a function will always be represented in tabular form.

The MBO<sub>s</sub> of the output tables are the following:

Figure 5.6: MBO of the output tables



### 5.2.3 Load Arguments

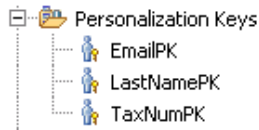
The input parameters are supplied by the user through a form in a screen of the application:

Figure 5.7: Search person BP screen

The screenshot shows a mobile application screen titled "Cerca Persona". At the top left is a "Cancel" button, and at the top right is a "Cerca" button. Below the title bar, the text "Inserisci i dati per ricercare la persona da visitare:" is displayed. There are three input fields: "Cognome", "Email", and "Cod. Fiscale". At the bottom of the screen is a large blue button labeled "Cerca".

The user-supplied values are stored in the Personalization Key, one for input parameter, specially created by the application developer:

Figure 5.8: Personalization Key



After that the Personalization Key Arguments are used as Load Arguments:

Figure 5.9: Load Arguments

Argument	Data Source			Value			
	Datatype	Nullable		Propagate to Attribute	Personalization Key	Synchronization Parame...	Default Value
I_COGNOME	STRING(35)				LastNamePK		<NULL>
I_ENVELOPE	ZUB_READ_...						{'DIVMERCATO'...
I_EMAIL	STRING(241)				EmailPK		<NULL>
I_TAXNUM	STRING(20)				TaxNumPK		<NULL>

Such Load Arguments allow you to download from the EIS only BP corresponding to the value provided by them. So can greatly reduce the flow of data from the EIS to the SUP.

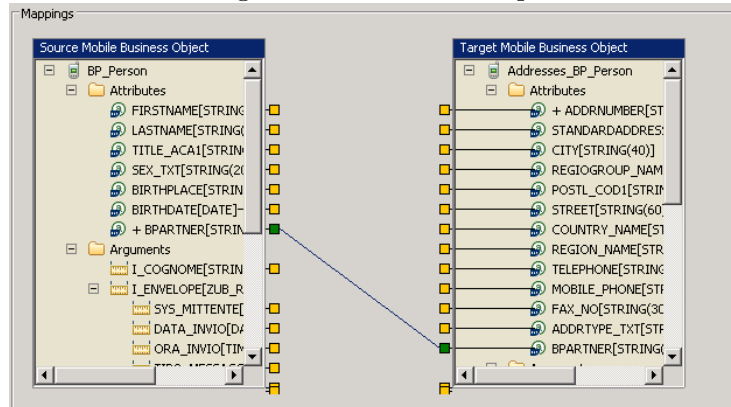
In addition, the Load Arguments allow you to partition the Unwired Server cache (CDB) to allow parallel refresh in each partition, increasing the performance of the system.

### 5.2.4 Relationship

The MBOs can be joined together by relationships, using as primary and secondary keys attributes and input parameters.

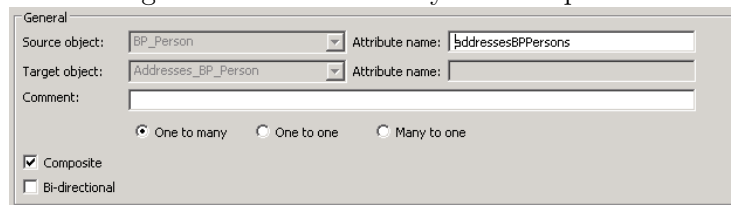
To create a relationship between the MBO BP\_Person and Addresses\_BP\_Person, binding the following attributes.

Figure 5.10: Relationships



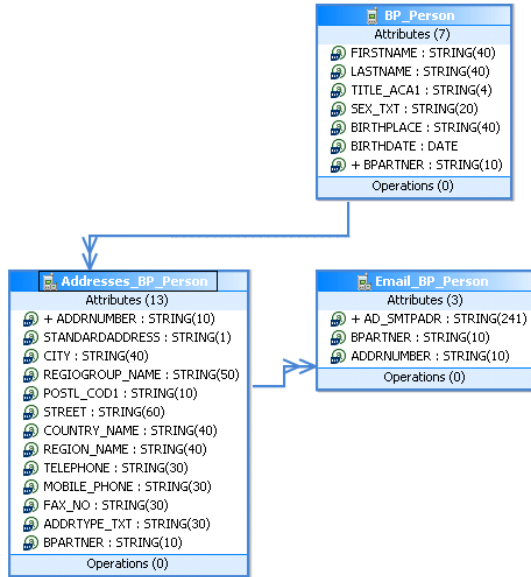
By setting the relationship as one to many and composite, so as to allow the changes to MBOs parents, are propagated to the children.

Figure 5.11: One-to-many and composite



Creating a relationship also between the MBO Adresses\_BP\_Person and Email\_BP\_Person, the MBO related to the BP person are so closely linked:

Figure 5.12: Relationship between MBO

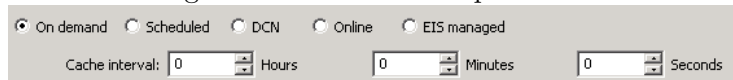


### 5.3 Cache Group and Synchronization Group

In this section we will discuss how it is implemented the Data Refresh between the Enterprise Information System (EIS) and the Sybase Un-wired cache (CDB), and the Synchronization between the mobile devices and the CDB.

The three MBO are derived from three output tables of a single function, so they must stay within the same cache group, in which we define the Data Refresh behavior.

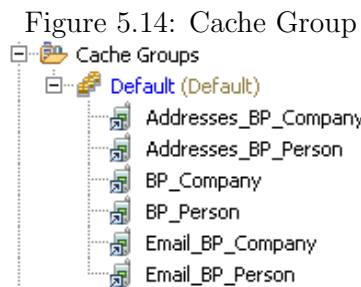
Figure 5.13: Cache Group Policies



We have chosen the Data Refresh schedule type, on demand policy. So

the refreshes with EIS can be done on user request and then reduce the workload of the EIS. We also notice that as the cache interval put all the parameters to 0, so when a user makes a request, every time the data will be requested from the EIS, thus keeping the data always consistent.

If we instead wanted to request a company BP, then we should call another function, ZUB\_READ\_BP\_2. In output we decide to return the same tables of ZUB\_READ\_BP\_1, except now the data are related to the company BP. Once you have created the MBO for these tables in output, we can note that will be included within the same cache group of the MBO of the function ZUB\_READ\_BP\_1.

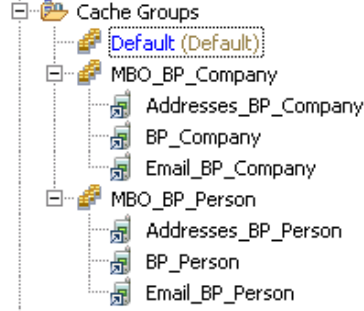


So if a Data Refresh is performed also for only one of these MBO, will be made a refresh also all other MBO within the same Cache Group, although not required. So when does a refresh for the MBO of person BP, a refresh automatically will be made for the MBO of the company BP. This leads to an overload of work and a greater flow of unnecessary data. To remedy this problem then the cache group is divided into two cache groups, one containing the MBO of the person BP and the other of company BP.

This phase of the subdivision of the cache group can also be made outside of the development phase. So if at some point the system administrator realizes that the cache group has too many MBO, may decide to split the cache group so as to reduce the workload and the flow of useless data. So we can see that the system, making it possible to divide the cache group, offers a **transparency of performance** in case of increased workload.

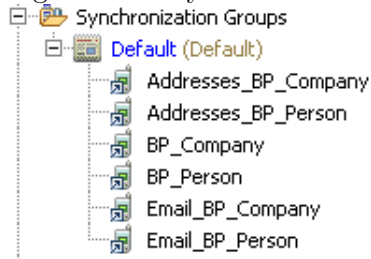


Figure 5.15: Cache Group divided



As regards synchronization, instead we can safely keep all MBO within a single Synchronization Group:

Figure 5.16: Synchronization



Which requires a synchronization between the mobile devices and Un-wired Server cache (CDB) every 10 minutes.

Figure 5.17: Synchronization interval



Since the data of MBO in question are not subject to major changes, if we keep them within a single Synchronization Group does not cause large workloads.

## 5.4 Workflow

Among the three functional parts of an application, *Workflow* represents the **presentation logic**, which resides within mobile devices *in the horizontal organization of the system*. It is deployed inside the *container* into the mobile device and it consists of the code HTML5, JavaScript and CSS of the *Hybrid Application* that implements *user view and controls*. It communicates with the container to make online requests or to exploit the potential of the mobile device as the camera. Between the layers of the architectural model of the system, it represents the highest layers, the **application layer**.

### 5.4.1 Code HTML5 and Javascript

The entire HTML5 code is contained within a file, *hybridapp.html*. It contains all the screens of the hybrid application, each within its own div element. For example, consider the initial screen of the application:

Figure 5.18: Start page



Its HTML5 code corresponds to the following div, inside the file *hybridapp.html*:

```

<div id="StartScreenDiv" sup_screen_title="Start"
    style="display: none"
    sup_menuitems="Chiudi App, Chiudi_App"
    sup_okaction="doCancelAction()"
    sup_autoarrangemenus="true">
<script>
    if (hwc.isIOS())
    {
        document.writeln(
            "<h3 id=\"StartScreenTitle\"
            class=\"screenTitle\">Start</h3>");
    }
</script>
<ul id="StartScreenDivMenu" class="menu">
    <li>
        <a class="nav" href="javascript:void(0)"
            name="Chiudi App" id="StartChiudi_App"
            onclick="menuItemCallbackStartChiudi_App(
                );">Chiudi App
        </a>
    </li>
</ul>
<form style="margin: 0px;" name="StartForm"
    id="StartForm" onSubmit="return false;"
    autocomplete="on"
    sup_show_alert_on_validation_error="true">
<table class="screen">
    <tr>
        <td colspan="2">
            <span id="StartForm_help"
                class="help">
            </span>
        </td>
    </tr>
    <tr>
        <td colspan="2" id="topOfStartForm"></td>
    </tr>
    <tr>
        <td colspan="2">
            <img src="" width="65%" height="25%"
                id="key95"

```

```

sup_static_options="true"
name="" alt=""/>
</td>
</tr>
</table>
<div id="key96" sup_html_type="htmlview">
    Benvenuti, <br/>
questa la vostra nuova applicazione CRM mobile. Seleziona
la tipologia di ricerca da fare:<br/><br/>
</div>
<table class="screen">
<tr>
<td colspan="2">
<!-- Note: using button tag seems doesn't
work for iOS when we have both text
and image , worked fine with BB6,
android and WM 7 -->
<button type="button"
data-role="none" ignored="true"
onclick="menuItemCallbackStartCerca_BP_persona();"
id="Cerca_BP_persona"
value="Cerca BP persona"
label-position="left" >Cerca BP
persona
</button>
</td>
</tr>
<tr>
<td colspan="2">
<!-- Note: using button tag seems doesn't
work for iOS when we have both text
and image , worked fine with BB6,
android and WM 7 -->
<button type="button"
data-role="none" ignored="true"
onclick="menuItemCallbackStartCerca_BP_azienda();"
id="Cerca_BP_azienda"
value="Cerca BP azienda"
label-position="left" >Cerca
BP azienda
</button>
</td>

```

```

        </tr>

        <tr><td colspan="2" id="bottomOfStartForm"></td></tr></table>
    </form>
</div> <!-- end of screen Start -->

```

As we can see in the code there are calls to JavaScript functions, defined and implemented within the *HybridApp.js* file. This file contains functions for common menu, screen and database operations, that is, all those operations to be performed, for example, when you type a button on the screen or a menu option, or when an online request is made to SUP.

From the HTML5 code in the *hybridapp.html* file, we can see that when I type the menu button “Chiudi”, it is called the function *menuItemCallbackStartChiudi\_App()*. This function is resident in the *HybridApp.js* file and has the following code:

```

function menuItemCallbackStartChiudi_App() {
    if (!hwc.customBeforeMenuItemClick('Start', 'Chiudi_App'))
    {
        return;
    }
    hwc.closeWorkflow();
    hwc.customAfterMenuItemClick('Start', 'Chiudi_App');
}

```

Such function via the function *hwc.closeWorkflow()* causes the closure of the workflow, and then the application.

The function *hwc.closeWorkflow()* is contained within the *API.js* file which is a library of JavaScript functions that allow you to communicate with the *Hybrid Web Container*. Then the function *hwc.closeWorkflow()* does nothing more than communicate with the container to close the application.

Here we have a first taste of how the code HTML5/JS of a hybrid application communicates with the Hybrid Web Container within any mobile platform, using JavaScript functions of a library previously provided.

Thus it becomes evident that the container in mobile devices is the *mid-*

*deware layer*, allowing the workflow, representing the *application layer*, to carry out any operation, such as the closure of the workflow itself, with a JavaScript function, regardless of the platform on which the workflow runs, providing **platform and programming language transparency**.

Taking another example, we can see that in the code of the function `menuItemCallbackStartChiudi_App()` another function is called, `hwc.customBeforeMenuItemClick()`, which resides within the `Custom.js` file, containing all the functions to be performed before and after the happening of any event, such as typing any button or loading of any page.

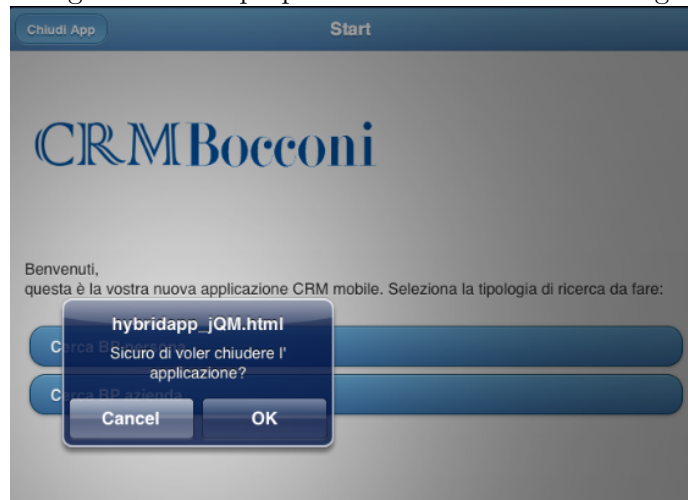
So before performing the closing operation performed by the function `menuItemCallbackStartChiudi_App()`, following the pressing of any button having label “Chiudi”, the function performed is `hwc.customBeforeMenuItemClick()`.

Usually all functions within `Custom.js` are empty, it is the application developer to implement them to customize the graphical interface of the application. So if the developer wanted to bring up a confirmation closing popup when typing the “Chiudi” button present on any window, the code of the `hwc.customBeforeMenuItemClick()` is as follows:

```
hwc.customBeforeMenuItemClick = function(screen , menuItem) {
    if(menuItem == 'Chiudi_App'){
        return hwc.showConfirmDialog(" Sicuro di voler
            chiudere l' applicazione?",
            "Chiudi applicazione");
    }
    return true;
};
```

where with the function `hwc.showConfirmDialog()`, we are able to bring up a confirmation popup:

Figure 5.19: Pop-up confirmation window closing



If we were to write this application in native code, the operation display of the popup would require a specific function for the mobile platform that would run the application. In this case, thanks to the container is not necessary, as it will be the same container to retrieve the native code for the appearance of the popup.

## 5.4.2 Online Request

What we want to do now is make an online request at the Enterprise Informatio System (EIS) to request the personal data of a BP person. The following HTML5 code in the hybridapp.html file is generated to implement the search page for a person BP:

```
<div id="Cerca_PersonaScreenDiv" sup_screen_title="Cerca
  Persona" style="display: none" sup_menuitems="Cerca ,Cerca"
  sup_okaction="doSaveActionWithoutReturn()"
  sup_autoarrangement="true">
  <script>if (hwc.isIOS()) {
    document.writeln(
      "<h3 id=\"Cerca_PersonaScreenTitle\"
```

```

                class="\screenTitle">Cerca Persona
            </h3>");
        }
</script>
<ul id="Cerca_PersonaScreenDivMenu" class="menu">
    <li><a class="nav" href="javascript:void(0)"
        name="Cerca" id="Cerca_PersonaCerca"
        onclick="menuItemCallbackCerca_PersonaCerca()
            ;">Cerca
        </a>
    </li>
    <li><a class="nav" href="javascript:void(0)"
        name="Cancel" id="Cerca_PersonaCancel"
        nclick="menuItemCallbackCerca_PersonaCancel
            ()">Cancel
        </a>
    </li>
</ul>
<form style="margin: 0px;" name="Cerca_PersonaForm"
    id="Cerca_PersonaForm"
    onSubmit="return false;" autocomplete="on"
    sup_show_alert_on_validation_error="true">
<table class="screen">
    <tr><td colspan="2">
        <span
            id="Cerca_PersonaForm_help" class="help">
        </span></td>
    </tr>
    <tr>
        <td colspan="2"
            id="topOfCerca_PersonaForm"></td>
    </tr>
</table>
<div id="key83" sup_html_type="htmlview">
    <h4>
        Inerisci i dati per ricercare la
        persona da visitare:
    </h4><br />
</div>
<table class="screen">
    <tr>
        <td class="left">

```



```

        <label for="CognomeKey">Cognome</
        label></td>
<td class="right">
    <input class="right" type="text"
        id="CognomeKey" sup_html_type="text"
        sup_max_length="32767"
        sup_num_of_decimals="0"/>
    <span id="Cerca_Persona_CognomeKey_help"
        class="help">
    </span>
</td>
</tr>
<tr>
<td class="left">
    <label for="EmailKey">Email</label></
    td>
<td class="right">
    <input class="right" type="text"
        id="EmailKey" sup_html_type="text"
        sup_max_length="32767"
        sup_num_of_decimals="0"/>
    <span id="Cerca_Persona_EmailKey_help"
        class="help">
    </span>
</td>
</tr>
<tr>
<td class="left">
    <label
        for="Codice_Fiscale_Key">Cod. Fiscale
    </label>
</td>
<td class="right">
    <input class="right" type="text"
        id="Codice_Fiscale_Key"
        sup_html_type="text"
        sup_max_length="32767"
        sup_num_of_decimals="0"/>
    <span id="
        Cerca_Persona_Codice_Fiscale_Key_help
        "
        class="help">

```

```

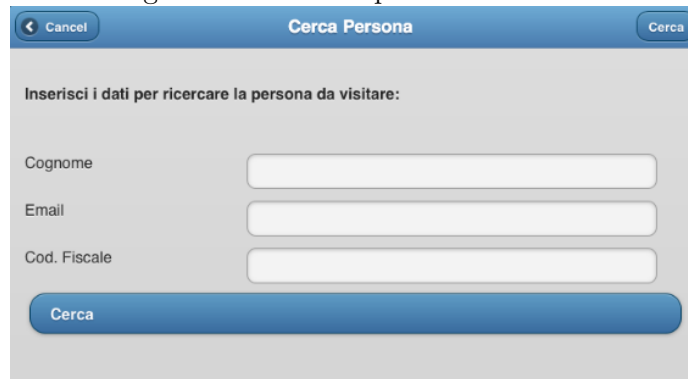
        </span>
    </td>
</tr>
<tr>
    <td colspan="2">
        <!-- Note: using button tag seems doesn't
            work for iOS when we have both text
            and image , worked fine with BB6,
            android and WM 7 -->
<button type="button" data-role="none"
        ignored="true"
        onclick="menuItemCallbackCerca_PersonaCerca()
            ;"
        id="Cerca" value="Cerca"
        label-position="left" >
        Cerca
    </button>
    </td>
</tr>

<tr>
    <td colspan="2"
        id="bottomOfCerca_PersonaForm">
    </td>
</tr>
</table>
</form>
</div> <!-- end of screen Cerca Persona -->

```

The result of which is the following page:

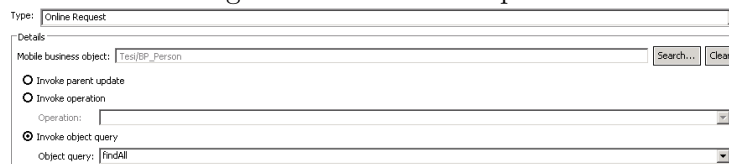
Figure 5.20: Search person BP screen



Each textbox in the code corresponds to the HTML5 tags `<input>`. The values entered in a textbox is stored within a key. For example in the case of the textbox “Cognome”, the key corresponds to `CognomeKey`, for “Email” the key is called `EmailKey`, while for the “Codice Fiscale” `Codice_Fiscale_Key`. Such keys in HTML5 code are indicated inside the `<input>` tag, as the attribute `ID` of the tag.

The online request to obtain the data of the BP person, must be forwarded to the Mobile Business Object (MBO) implemented specifically to require such data to the EIS. The MBO in question, as seen in the previous sections is the `BP_Person`.

Figure 5.21: Online Request



When the data arrives at the mobile device, are analyzed through a query, `findAll()`, which scans line by line of the output table.

When we created the MBO BP\_Person, we bound each Load Arguments with a Personalization Key, so you can filter the amount of data downloaded from the EIS, and get only the person BP corresponding to the value of “Cognome”, “Email” and “Codice Fiscale” provided by the user. So you have to pass these values to the corresponding Personalization Key. This is done by binding the Personalization Key to the keys of the textbox containing the values entered by the user.

Figure 5.22: Personalization Key mapping

Personalization Key Mapping

Personalization Key	Key	Project
LastNamePK	CognomeKey	Tesi
EmailPK	EmailKey	Tesi
TaxNumPK	Codice_Fiscale_KEY	Tesi

So when the user types “Monti” in the textbox “Cognome”, then the mobile device will receive from the EIS only the person BP with the surname “Monti”.

Figure 5.23: List of the person BP with surname Monti



Going into more detail of the BP Ambrogio Monti:

Figure 5.24: Details of person BP Ambrogio Monti

Field	Value
BPartner	0000442391
Nome	AMBROGIO
Cognome	MONTI
Sesso	Maschio
Luogo di nascita	
Data di nascita	09/mag/1927

Buttons: Elenco Indirizzi, Codice Fiscale, Leads

So we have seen how to implement an online request, that is as an application on a mobile device makes a request to the EIS, through the MBO within the SUP. In addition we have also seen how, through the keys of the textbox mapped to Personalization Key, the user provides the input parameters to the function ZUB\_READ\_BP\_1 that the EIS must perform to meet and exceed the request.

When you push the button “Cerca”, within the search page of a person BP, you call a JavaScript function, *menuItemCallbackCerca\_PersonaCerca()* in the file HybridApp.js, making the online request:

```
function menuItemCallbackCerca_PersonaCerca () {
    if (!hwc.customBeforeMenuItemClick('Cerca_Persona', '
    Cerca')) {
        return;
    }
    var rmiKeys = [];
    var rmiKeyTypes = [];
    var rmiInputOnlyKeys = [];
    var rmiInputOnlyKeyTypes = [];
    rmiKeys[0] = 'CognomeKey';
```

```

rmiKeyTypes[0] = 'TEXT';
rmiKeys[1] = 'EmailKey';
rmiKeyTypes[1] = 'TEXT';
rmiKeys[2] = 'Codice_Fiscale_Key';
rmiKeyTypes[2] = 'TEXT';
rmiInputOnlyKeys[0] = 'CognomeKey';
rmiInputOnlyKeyTypes[0] = 'TEXT';
rmiInputOnlyKeys[1] = 'EmailKey';
rmiInputOnlyKeyTypes[1] = 'TEXT';
rmiInputOnlyKeys[2] = 'Codice_Fiscale_Key';
rmiInputOnlyKeyTypes[2] = 'TEXT';
var dataMessageToSend = hwc.
    getMessageValueCollectionForOnlineRequest(
        Cerca.Persona, 'Cerca', rmiKeys, rmiKeyTypes);
var inputOnlyDataMessageToSend = hwc.
    getMessageValueCollectionForOnlineRequest(
        Cerca.Persona, 'Cerca', rmiInputOnlyKeys,
        rmiInputOnlyKeyTypes);
if (hwc.validateScreen('Cerca.Persona',
    hwc.getCurrentMessageValueCollection(),
    rmiKeys) && hwc.saveScreens(true))
{
    hwc.doOnlineRequest('Cerca.Persona', 'Cerca',
        100, 0, '', null, dataMessageToSend,
        inputOnlyDataMessageToSend.serializeToString());
}
hwc.customAfterMenuItemClick('Cerca.Persona', 'Cerca');
}

```

The JavaScript code is interpreted by the Hybrid Web Container and translated into native code, so as to make the online request. In this case, since the step is an online request, it becomes more obvious as the container, which is the middleware layer, as well as offering the transparency of platform and programming language, as seen in the previous chapter, provides **location and protocol transparency**. As the implementer of the JavaScript function does not take into account where are the SUP and what type of protocol to be used to achieve it.



# Chapter 6

## Conclusions

After the study of the entire documentation of the Sybase Unwired Platform (SUP) to learn about its capabilities, how it works, how to build with it a hybrid application and after creating an application, see the finished product was very rewarding.

This experience has allowed me to combine many notions that before I had only studied in university books and that seemed distant and independent of each other: from pure theoretical concepts in my mind, they have taken shape, connecting to each other, until the implementation and use of the application.

As you can see from reading the document, the Sybase Unwired Platform is the component that provides the transparency necessary for a distributed system. There are many techniques to solve the problems inherent in a distributed system, and with this study we were able to find out which of these are used by the Sybase Unwired Platform.

What struck me the most was the ease with which the SUP allows you to create hybrid applications and communication with different types of back end platforms, through the realization of the middleware layer, which in the SUP is called Mobile Business Object.

Once you have made the hybrid application, I could see that it was less fluid of a native application. And the only way to improve this is to



increase the computing power of the mobile device. On this front, then there should be no major concerns: smartphones are becoming more and more powerful, doing so will make such applications equivalent to the native ones.

Moreover from later versions of the SUP, the implementation is made even easier. By chance, also, to translate the HTML5 code, JavaScript and CSS in native code, allowing greater flexibility in the implementation of the application.

In many cases, the transfer of resources from the Enterprise Information System (EIS) is still too slow. A future development in fact is to not install the Sybase Unwired Platform in a server cluster of the company, but to make available a cloud system, in order to increase the computing power of the entire distributed system.

Also others communication systems will be use, such as XML, JSON, and OData, for bi-directional communication of the user interface and the data between the mobile devices and the cloud, so to make lighter and faster data transfer and more simple and flexible implementation of the graphical interface and the mobile application in whole. SAP is investing heavily in its mobile platform to make it more efficient, and simple to implement applications.

# Acknowledgements

First I wanted to thank those who have contributed directly and indirectly to the development of this document. First my parents, who have allowed me to begin the path of the academic studies, the conclusion of which is represented by this document. And my sister who shared with me his university career.

And then I would like to thank my friend Cesco, who just heard the proposal for an internship in Altevie , contacted me immediately , allowing me to start the internship . The colleague Marco Favero , with whom I had the first interview and the first contact in Altevie . The company tutor Francesco Garbellotto who followed me despite his busy schedule , colleagues Manuel Xicato and Diandra Morello I pestered with many questions, but always available to answer , even in spite of their numerous work commitments . My colleague Matteo Bonas who gave me a big help with the ABAP code and the CRM platform . My advisor Renzo Orsini and my university tutor for the internship Michele Bugliesi , who helped me in transforming the Sybase Unwired Platform in a possible thesis. And finally, but not least, my friend Cristi that willing helped me in the creation of many of the images in this document.

But I would also like to thank all those who have accompanied me on this journey university, which was one of the best times of my life. Starting with my colleague and friend Vito with which I started, and now I am going to conclude, sharing the difficulties and satisfactions of this route. But also Max and Jona, always ready to help me and partying. Marco, who has spent with me last summer on the books, and also the compan-

ions of projects and breaks during the long days of study, such as Davide, Kele and Popa.

And last, but not less important, on the contrary, I would like to thank all the friends who have shared with me long days in the library Civica in Mestre and Marghera, making me the way to this my personal results an experience lighter and fun, beginning by Saretta, faithful companion of study, but also Fava, Silvieta, Chiaretta, Kekka, Wewa, Annaki, Berto, Michi. Not to mention that Toni several times gave me to eat before the afternoon study.

# Bibliography

- [1] George Coulouris, Jean Dollimore, Tim Kindberg, Gordon Blair  
*Distributed Systems Concepts and Design.*  
Pearson Prentice Hall  
Fifth Edition  
2012.
  
- [2] Andrew S. Tanenbaum, Maarten Van Steen  
*Distributed Systems Principles and Paradigms.*  
Pearson Prentice Hall  
2nd Edition  
2006.
  
- [3] Neeta Deshpande, Snehal Kamalapur  
*Distributed Systems.*  
Technical Publications Pune  
2nd Edition  
2008.
  
- [4] Ajay D. Kshemkalyani, Mukesh, Singhal  
*Distributed Computing Principles, algorithms, and systems.*  
Cambridge University Press  
1st Edition  
2011.
  
- [5] Andrew S. Tanenbaum  
*Reti di calcolatori.*  
Pearson Prentice Hall

4th Edition  
2008.

- [6] John M. Wargo  
*PhoneGap Essentials Building Cross-Platform Mobile Apps.*  
Addison-Wesley Pearson Education  
1st Edition  
2012.
- [7] Lee S.Barney  
*Developing Hybrid Applications for the iPhone: Using HTML, CSS,  
and JavaScript to Build Dynamic Apps for the iPhone.*  
Addison-Wesley Pearson Education  
1st Edition  
2009.
- [8] Jamie Munro  
*20 Recipes for Programming PhoneGap: Cross-Platform Mobile De-  
velopment for Android and iPhone.*  
OReilly Media  
1st Edition  
2012.
- [9] Nizamettin Gok, Nitin Khanna  
*Building Hybrid Android Apps with Java and JavaScript.*  
OReilly Media  
1st Edition  
2013.
- [10] Silbershatz, Galvin, Gagne  
*Sistemi Operativi Concetti ed esempi.*  
Pearson Addison Wesley  
7th Edition  
2013.
- [11] Antonio Albano, Giorgio Ghelli, Rennzo Orsini  
*Fondamenti di basi di dati.*

Zanichelli  
2nd Edition  
2005.

- [12] Antonio Albano  
*Costruire Sistemi Per Basi Di Dati.*  
Addison-Wesley Pearson Education  
1st Edition  
2001.
- [13] Elliotte Rusty Harold  
*XML 1.1 Bible.*  
Wiley  
3rd Edition  
2004.
- [14] Shelly Powers  
*Learning JavaScript.*  
OReilly Media  
1st Edition  
2006.
- [15] Bruce Lawson and Remy Sharp  
*Introducing HTML5.*  
New Riders  
1st Edition  
2011.
- [16] SAP Mobile Platform  
*Sybase Unwired Platform 2.1 ESD #3: Fundamentals.*  
Sybase Product Documentation  
<http://infocenter.sybase.com/help/index.jsp?docset=/com.sybase.infocenter.pubs.docset-SUP-2.1.3/doc/html/title.html&docSetID=1838>  
2012.
- [17] SAP Mobile Platform  
*Sybase Unwired Platform 2.1 ESD #3: Installation Guide for*

- Runtime.*  
Sybase Product Documentation  
<http://infocenter.sybase.com/help/index.jsp?docset=/com.sybase.infocenter.pubs.docset-SUP-2.1.3/doc/html/title.html&docSetID=1838>  
2012.
- [18] SAP Mobile Platform  
*Sybase Unwired Platform 2.1 ESD #3: Sybase Unwired Workspace Mobile Business Object Development.*  
Sybase Product Documentation  
<http://infocenter.sybase.com/help/index.jsp?docset=/com.sybase.infocenter.pubs.docset-SUP-2.1.3/doc/html/title.html&docSetID=1838>  
2012.
- [19] SAP Mobile Platform  
*Sybase Unwired Platform 2.1 ESD #3: Mobile Data Models Using Mobile Business Objects.*  
Sybase Product Documentation  
<http://infocenter.sybase.com/help/index.jsp?docset=/com.sybase.infocenter.pubs.docset-SUP-2.1.3/doc/html/title.html&docSetID=1838>  
2012.
- [20] SAP Mobile Platform  
*Sybase Unwired Platform 2.1 ESD #3: Sybase Control Center for Sybase Unwired Platform.*  
Sybase Product Documentation  
<http://infocenter.sybase.com/help/index.jsp?docset=/com.sybase.infocenter.pubs.docset-SUP-2.1.3/doc/html/title.html&docSetID=1838>  
2012.
- [21] SAP Mobile Platform  
*Sybase Unwired Platform 2.1 ESD #3: System Administration.*  
Sybase Product Documentation  
<http://infocenter.sybase.com/help/index.jsp?docset=/com.sybase.infocenter.pubs.docset-SUP-2.1.3/doc/html/title.html&docSetID=1838>  
2012.

- [22] SAP Mobile Platform  
*Sybase Unwired Platform 2.1 ESD #3: Developer Guide Mobile Workflow Package.*  
Sybase Product Documentation  
<http://infocenter.sybase.com/help/index.jsp?docset=/com.sybase.infocenter.pubs.docset-SUP-2.1.3/doc/html/title.html&docSetID=1838>  
2012.
- [23] SAP Mobile Platform  
*Sybase Unwired Platform 2.1 ESD #3: Sybase Unwired WorkSpace Mobile Workflow Package Development.*  
Sybase Product Documentation  
<http://infocenter.sybase.com/help/index.jsp?docset=/com.sybase.infocenter.pubs.docset-SUP-2.1.3/doc/html/title.html&docSetID=1838>  
2012.
- [24] SAP Mobile Platform  
*SAP Mobile Platform 2.3 Developer Guide: Hybrid Apps.*  
Sybase Product Documentation  
<http://infocenter.sybase.com/help/index.jsp?docset=/com.sybase.infocenter.pubs.docset-SMP-2.3.0/doc/html/title.html&docSetID=1939>  
2012.
- [25] SAP Mobile Platform  
*SAP Mobile Platform 2.3 SAP Mobile WorkSpace Hybrid App Package Development.*  
Sybase Product Documentation  
<http://infocenter.sybase.com/help/index.jsp?docset=/com.sybase.infocenter.pubs.docset-SMP-2.3.0/doc/html/title.html&docSetID=1939>  
2013.