WIRELESS INTERNET SOFTWARE ENGINEERING IST-2000-30028

Title:
Architecture Handbook

Version: 1.06
Date : 21 Apr 04
Pages :

Author(s):
J. Kalaoja; E. Niemelä; A. Tikkala; P. Kallio; T. Ihme; M. Torchiano

To:
WISE CONSORTIUM

Printed on:
20-May-04 9:51

Status:

[ ]	Draft
[ ]	To be reviewed
[x]	Proposal
[ ]	Final / Released

Confidentiality:

[ ]	Public	- Intended for public use
[X]	Restricted	- Intended for WISE consortium only
[ ]	Confidential	- Intended for individual partner only

Deliverable ID: D4 (Part D)

Title:

Architecture Handbook

Summary / Contents:

This document is a part of the deliverable D4 produced in the task 2.1 of the Wise project. Deliverable D4 includes four parts: Part A: Architectural guidelines, Part B: the WISA (Wireless Internet Service Architecture) architectural knowledge base and its reference architecture (WISA/RA), Part C: Analysis of the pilot architectures, and Part D: Handbook of reusable architectural assets.

This document contains a set of tools that can be used to build wireless services. This document should be read after the knowledge contained in D4B has been assimilated.

The document provides three types of reusable architectural assets: 1) typical architectures that can be used as starting points to develop wireless service architectures; 2) architectural styles and patterns that can be used to develop services, and 3) existing services that can be re-used in new services.

TABLE OF CONTENTS

CHANGE LOG

Vers.	Date	Author	Description	Comments
1.00	20 June 03	M.Torchiano	Created from parts of previous D4B (made by J. Kalaoja, E. Niemelä, A.Tikkala, P. Kallio, T.Ihme and M. Torchiano)	
1.01	17 July 03	T.Ihme	Introduced chapter on architectural styles and patterns	
1.02	29 August 03	T.Ihme	Expanded chapter on patterns	
1.03	22 Sep. 03	M.Torchiano	Added typical architectures and wireless patterns	
1.04	5 Oct 03	M.Torchiano	Refined typical architectures and patterns	
1.05	6 Oct 03	M.Torchiano	Integrated several contributions	

APPLICABLE DOCUMENT LIST

Ref.	Title, author, source, date, status	Identification
1	D4A - Achitectural guidelines	v. 1.04
2	D4B - WISA reference architecture	v. 1.04
3	D4C - Analysis of pilot architectures	v. 1.04

1. INTRODUCTION

This document presents WISE architectural handbook for wireless Internet services. The handbook is intended to be a sort of vademecum of the wireless service developers, providing him/her with a set of solutions to recurrent issues.

The main stakeholder of this document is the "new" and possibly inexperienced wireless service developer, who has some knowledge about software development (which we do assume is deep) and a limited if any knowledge about the wireless services domain.

This document is a part of the D4 deliverable that as a whole provides a set of assets to be used in wireless service engineering. Part A, Architectural guidelines (see ref. 1), defines 1) the terminology, 2) viewpoints and 3) notation appropriate in the development of wireless services. These guidelines have also been applied in the documentation of WISA/RA and the basic services in this document but only from the point of view of the users, not the developers, of the reference architecture and its services. Part B contains the reference architecture and its constituents; the purpose of D4B is to provide the basic understanding of the issues of architecting a wireless service and provide the basic knowledge and terminology to understand the new domain. Part C, Analysis of pilot architectures, gives valuable feedback of the use of the architectural guidelines and WISA/RA. The purpose of the Part C is also to encourage the architects to analyze architecture before its use because it leads to better quality of services and decrease development cost. In summary, these three parts of D4 provide a set of reusable assets for wireless service development and all of them are encouraged to be used in order to maximize benefits from the use of WISA knowledge base.

The content of this document is a set of reusable architectural assets:

- Typical examples of wireless services architectures that can be used as starting point to develop custom solutions.
- Architectural styles and patterns to be reused.
- Descriptions of services. The purpose of the descriptions is to assist service developers to use services as building blocks in the development of wireless services. Therefore, the emphasis was put on the quality, features and interfaces a service provides to its users, not its internal functional properties.

1.1 TYPICAL PROBLEMS IN WIRELESS SERVICES

The novice wireless internet service developer faces many problem when engineering and designing an application destined to provide a service in the wireless domain. One of the objectives of this architectural handbook is to provide an overview of the most common issues encountered in the wireless service domain. When we will present the reusable assets (typical architectures, patterns, styles and existing services) we will show how they address the main wireless issues.
We conducted an investigation on the wireless specific issues and problems, we found that the main features that characterize wireless services are:

- Heterogeneous clients
- Limited device capabilities
  - Screen size
  - Memory size
  - Performance
  - Power supply
- Limited bandwidth
- Discontinuous network connection
  - Intermittent availability

- Mobility
    - Transition among "cells"
- Location awareness
- Gap between demonstrator & deployed service (surprise project)
- Several parties participating providing the services or its components
- dominant position of telecom operators when providing services
- different network technologies in different countries

It is possible to deal with the above issues at several levels in the development of wireless services. Some of them can be addressed by means of low-level solutions and patterns, while other can be dealt with by means of architectural solutions.

1.2 STRUCTURE AND CONTENT

The handbook is divided into three parts that describe the reusable assets corresponding to the three types of reusable assets. The next chapters present a collection of reusable assets:

- Typical architectures are described in Chapter 3
- The architectural styles and patterns are presented in Chapter 4
- The catalog of available services is in Chapter 5

2. ABBREVIATIONS

API	Application Programming Interface
ATM	Asynchronous Transfer Mode
BSS	Business Support Systems
BTS	Base Transceiver Station
C/S	Client Server
CORBA	Common Object Request Broker Architecture
COTS	Commercial Off-The-Shelf
CRM	Customer Relationship Management
DNS	Domain Name Server
EMS	Enterprise Messaging Server
FTP	File Transfer Protocol
GGSN	Gateway GPRS Support Node
GIS	Geographic Information Systems
GPS	Global Positioning System
GUI	Graphic User Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HUS	Heterogeneous User Interface Service
DSOM	Distributed System Object Model
IM/P	Instant Messaging and Presence service
ITF	Interface
J2EE	Java 2 Enterprise Edition
J2ME	Java 2 Micro Edition
MMS	Multi Media Messaging
MOTS	Modified Off-The-Shelf
MP3	MPEG1 Layer 3
MPEG	Moving Picture Experts Group
MVC	Model-View-Controller architectural pattern
NFR	Non-functional Requirements
OCM	Original Component Manufacturer
ODBC	Open DataBase Connectivity
OLE	Object Linking and Embedding
OS	Operating System
OSE	Open System Environment
OSI	Open Systems Interconnection Model
OSS	Operating Support Systems
P2P	Peer-to-Peer
PAC	Presentation-Abstraction-Control
PAs	Presence Agents
PC	Personal Computer
PDA	Personal Digital Assistant
PING	Packet INternet Groper
QoS	Quality of Service
RPC	Remote Procedure Call
RTP	Rapid Transport Protocol
RTSP	Real-Time Streaming Protocol
SIP	Session Initiation Protocol
SLA	Service Level Agreement
SMC	Service Management Component
SMS	Short Message Service
SQL	Structured Query Language

TCP/IP	Transmission Control Protocol/ Internet Protocol
TOM	Telecom Operations Management
UDP	User Datagram Protocol
UE	Universal Explorer
UIML	User Interface Markup Language
WAP	Wireless Application Protocol.
WISA	Wireless Internet Service Architecture
VM	Virtual Machine / memory
VP	Viewpoint
WWW	World Wide Web
VXML	Voice eXtensible Markup Language
XML	eXtensible Markup Language

3. TYPICAL ARCHITECTURES

When a software system developer (team) enters a new domain, several problems and issues are faced for the first time. First of all it is important to understand the new domain in terms of terminology, concepts and relationships among them. In particular it is useful to have a reference architecture that describes the functions and the corresponding decomposition. In fact each domain, and wireless Internet services are no exception, adopts a typical decomposition of functions and features. These details are provided by the D4 part A, which is a sort of introduction to the fundamental architectural concepts of the wireless Internet services domain.

Once the domain is known in its basic concepts the problem is where to start to develop a wireless service. While the reference architecture provides the generic guidelines to design the architecture, it is too abstract. Often the right point to start from is an example. The architectures presented in this chapter play exactly this role: they are examples of typical architectures of wireless services.

The novice wireless service developer can build a new service starting from one of these typical architectures. The description of the typical architectures presented in this chapter is kept at a fairly abstract level, avoiding application specific details that could hinder the comprehension. In addition keeping a high level of abstraction makes it possible to easily adapt the architecture to specific requirements.

The process we followed to find the typical architectures presented here consists in the following steps:

- identification of possible sources of architectures
- mining of the source to find suitable architectures
- abstraction of the architectures to purge application-specific details
- repackaging of the architectures using the WISE guidelines.

After a brief investigation we identified three main sources of information that could provide us with meaningful wireless services architectures, they are:

- WISE pilots
- External published projects
- Interviews

The pilots developed inside this project emerged immediately as good candidates to provide wireless services architectures that could be used as examples. A natural objection could be that they are prototypal applications and thus not meaningful in an industrial contexts. There are two answers to this objection. First, event though prototypal, they address typical real-world problem and are developed by the industry therefore their architectures are of interests. Second, there is anecdotal evidence that the vast majority of wireless services are first developed as quick prototypes and only after the first period of service they are engineered to achieve scalability and efficiency.

Another source of potential typical architecture is represented by other projects operating in the domain of wireless services. The architectures can be found by looking at the deliverables or dissemination documents produced by such projects.

Finally, typical architectures can be extracted from the results of interviews with wireless service developers. We considered the option of carrying a series of interviews with developers working in companies developing wireless services. In fact there are several services either deployed or under development.

In this version of the handbook we mined the architectures only from the pilots. In the next iteration we plan to exploit the other sources too.

The mining phase has the purpose of finding suitable architectures. The key point here lies in the definition of what is suitable. The approach we taken is based on subjective judgment. The directive adopted in such a judgment is to consider suitable an architecture that addresses a typical problem and potentially can be applied to several similar cases.
In the case of the pilots, since they were chosen for their representativeness, the pilots' architectures match the above directive.

The abstraction phase aims at removing all the application specific details. An important decision, in this phase, has to be taken on which details are specific and which are generic. As a rule of thumb we state that application specific details are those that do not occur across similar services. The goal of this phase is to produce a stripped down architecture that both can be easily understood and is easy to customize.

Finally, it is important to describe the architectures in a uniform way, both in terms of content and notation. In principle the architecture can be found originally in very different forms. In this phase we describe them using the notation defined in D4 part A and the guidelines provided in D4 part B.

3.1 TA1: ADAPTATION OF WEB-BASED CONTENT PROVISIONING

This typical architecture describes an approach that can be used to adapt existing web-based applications to the wireless services context. This typical architecture is particularly aimed at content provisioning services, i.e. applications that allow browsing of information and reading of news. The main differences between a web-based application and its wireless counterpart consist in lower bandwidth available and reduced terminal presentation capabilities (mainly reduced display size and graphical functions).
Typical examples of such services are: stock quote monitoring, news reading, etc.

The main issues that characterize this architecture are:
- limited bandwidth,
- device limitation.

The user terminal must receive information which may be complex and are updated often; this requires a certain amount of bandwidth, and therefore we ought to adopt an appropriate protocol. In addition the presentation of information may require graphical capabilities and processing power on the user terminal side.
The essential feature of this typical architecture is the presence of a significant amount of information that is updated asynchronously, this updates must be conveyed to the users of the system so they can rely on recent information.

The application can provide information using the WAP protocol, which can be implemented in parallel to the existing HTTP. This approach allows reusing most of the system and developing only the presentation part. Since WAP and WML are simplified versions of HTTP and HTML respectively, the development of the new presentation part can be derived from the existing one.

This type of service is based on an asymmetric protocol, i.e. the client decides autonomously when to read information from the server. Thus the server must collect all the updates and feed them to the client when they connect. On their turn, clients have to connect periodically to load the updated information.
In this process an essential role is played by the cache database. Its purpose is to keep an up-to-date copy of the information that is required by the users.

The limited bandwidth issue is solved by using a "slimmer" protocol than HTTP that is WAP.
The device limitation is addressed leveraging the WML browser built-in in the device.

3.1.1 Conceptual Architecture

The following subsections describe the conceptual pilot architecture from different perspectives (or architectural views) according to D4 viewpoints.

3.1.1.1 Structural View

3.1.1.1.1 System Context

The system operates as an extension of a pre-existing web-based system. These two applications share several components as shown in Figure 1. The web server in addition to serving HTML pages must be able to serve WML pages through the WAP protocol which is managed by the gateway.
The wireless version can use the same caching database as the wireline version.

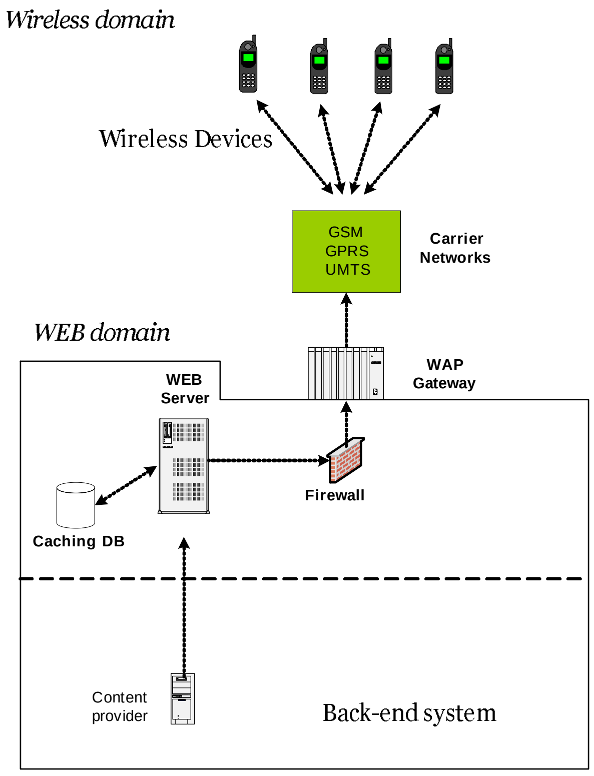


Figure 1. Overview of the execution environment.

3.1.1.1.2 Conceptual Structure

The functional conceptual structure is presented in Figure 2. The service provided actually consists in presentation and administration. The information presented is provided by a domain specific content provider service, which may have several different interfaces. The integration of the presentation and back-end features is achieved through a caching and adaptation service. Both the mobile and fixed users make use of the same User Service functionality.

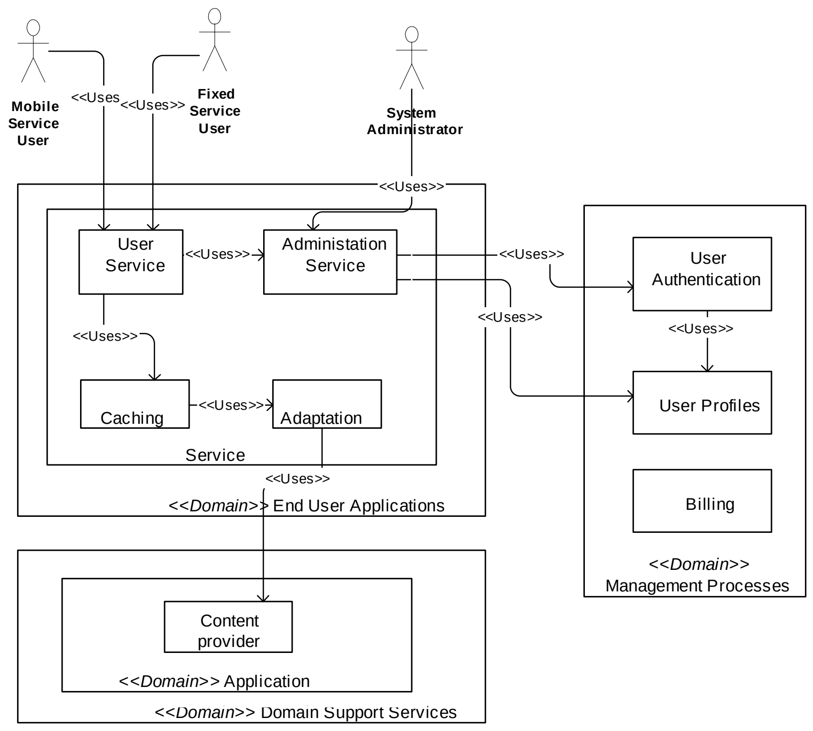


Figure 2 : Conceptual structure

The descriptions of actors are in Table 1 and the responsibilities of conceptual elements in Table 2.

Table 1: Actors

Conceptual Element	Description
Mobile Service User	Uses the service from mobile device.
Fixed Service User	Uses the service from a web browser.
System Administrator	Manages access rights.

Table 2: Responsibilities of conceptual elements.

Conceptual Element	Responsibility
User Service	Provides the presentation of information for both mobile and fixed users.
Caching	Keep an up-to-date copy of the information required by the users to improve performance
Adaptation	Receives and adapts the information produced by the back end
User Authentication	Takes care of authentication, security and user classes.
User Profiles	Stores user profile information.
Back-end Service	Provides the content to be feed to the users.

3.1.1.2 Behavioural View

Behavioural views are based on the main use cases of the system and presented as UML collaboration diagram. For this typical architecture these views are not relevant.

3.1.1.3 Deployment View

The deployment of conceptual entities to system nodes is depicted in Figure 3. Both mobile and fixed network devices connect to unique access node, the former through a WAP gateway and the internet, the latter directly through internet. The access node host the presentation-related user service, this node can be separated from the server node hosting the caching and adaptation service together with other supporting services. Finally the content provider sits on a separate node.

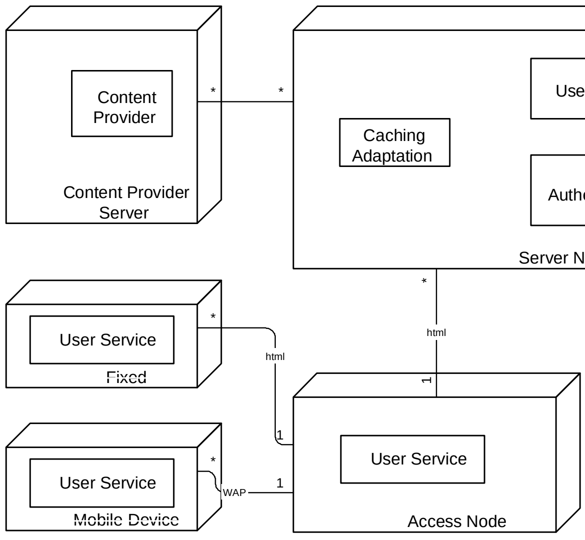


Figure 3: Conceptual Deployment

3.1.1.4 Development View

3.1.1.4.1 Business Context

The business model for this typical architecture is depicted in Figure 9.
Business roles in dark play some task in the operation of the service (role Network Operator is optional and not investigated in the Wise context). This task can either involve service provisioning (see those roles inside the dashed box) if there is some software components deployed in a networked structure, or not involve service provisioning (see roles outside the dashed box) if they have a business relationship prior to service provisioning (e.g. Application providers).

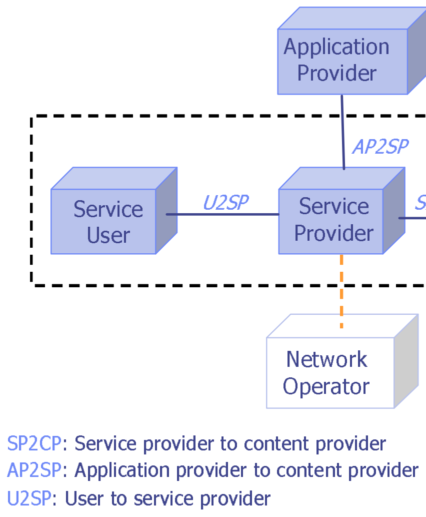

Application Provider
AP2SP
Service User
U2SP
Service Provider
SP2CP
Content Provider
Network Operator
SP2CP: Service provider to content provider
AP2SP: Application provider to content provider
U2SP: User to service provider

Figure 4. The business model

There are four main roles involved in the business model underpinning this typical architecture. The *Application Provider* delivers the application(s) used to present the information, cache the content, and to receive and adapt the content coming from the source. The *Content Provider* is the source of the information that will be presented to the user, who plays the role named *Service User*. All the roles defined in the business model can be played by multiple actors; of course there will be many users, but it is also possible to have several sources of information and different applications. Table 3 explains in more detail the monetary and information flows of the Figure 9.

Table 3: Monetary and information flows between the business actors.

Business flow	Type of the flow	Explanation
Service Provider to Service User	Content	The service provider provides the information required and updates it as it changes
Service User to Service Provider	Money	The user pay the service provider to get the service
Content Provider to Service Provider	Quotes Updates	The service provider receives the updates form the content provider.
Application Provider to Service Provider	Application (+ use license)	The application provider provides the application that manipulates the content to the service provider
Service Provider to Content Provider	Money	The service provider pays for the application

3.2 TA2: RICH INTERACTION

This typical architecture provides support for services featuring rich interaction and communication intensive (e.g. interactive multiplayer games) using thick client s. We consider thick clients that have some computing power (e.g. Java ME) including both smart-phones and wireless-enabled PDAs.
This typical architecture abstracts the essential features common to a category of wireless services. This category is characterized by a number of clients interacting with a server that is responsible for updating the shared state and keeping the clients consistent.
Typical examples of such services are: interactive multiplayer games, fleet position monitoring, etc.

While the previous typical architecture focused on a significant amount of that needs to be updated regularly, this architecture focus more on a timely and frequent update of small quantities of data and the consistency among the information presented to the users.

The main issues addressed by this category of wireless systems are:

- Synchronization between clients and server
- Bandwidth limitation
- Device capability limitation

Rich interactions require a frequent synchronization between the server and the user terminals; the server has to send the terminal updated information. As a consequence there is the need for a high bandwidth, which is in contrast with the limited availability in wireless networks. In addition the user terminal should be able to present a complex and evolving set of information.

The contrasting issues of frequent synchronization of the clients and limited bandwidth availability are solved by an ad-hoc protocol based on UDP that has a limited overhead. As a result the latency is within the acceptable limits.
The limited device capabilities are solved using an ad hoc graphical interface, since the built-in WML capabilities were not sufficient.

3.2.1 Conceptual Architecture

The following subsections describe the conceptual pilot architecture from different perspectives (or architectural views) according to D4 viewpoints.

3.2.1.1 Structural View

3.2.1.1.1 System Context

The networked environment for the application is presented in Figure 5. Clients have access to a GPRS network (UTMS in the future) which is connected to Internet, by means of a GGS Node. Having this access to Internet, clients are able to connect the Server. The Server uses Management Services (such as authentication and authorisation) which are provided by a host running on a node on the network (either in the same LAN or remote).

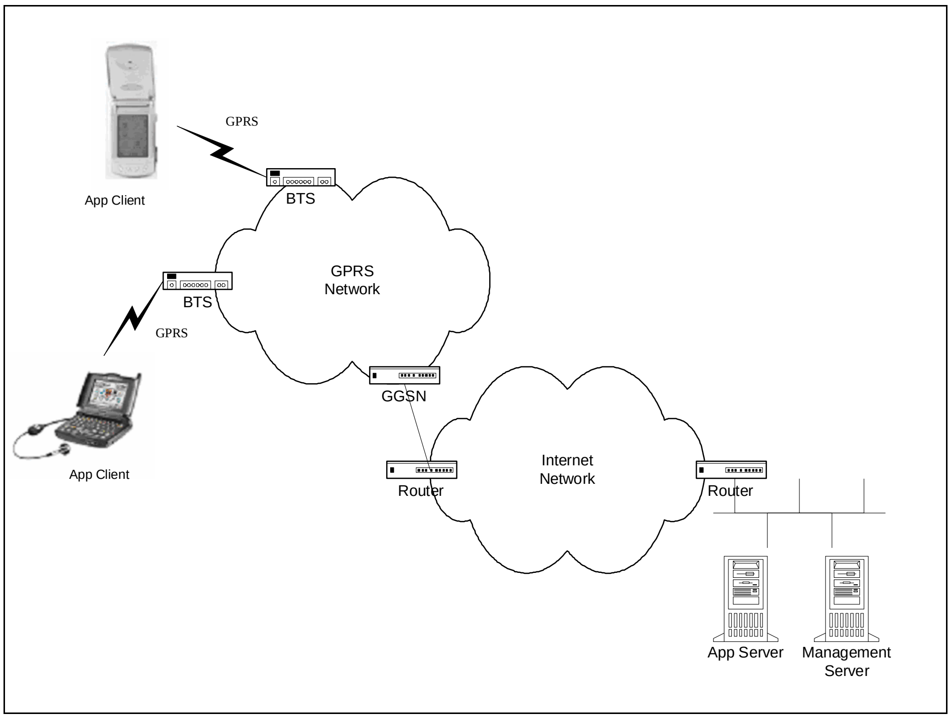


Figure 5. Overview of the execution environment.

3.2.1.1.2 Functional Structure

One of the main goals of selecting the conceptual entities was first to identify the generic application domain services that are common for different kind of computer games (and possibly other entertainment services). These generic services could be reused as a platform for different game applications. One of the main goals of Wise project is to provide a generic architecture for developing wireless services. The draft of conceptual structure is presented in Figure 6. The responsibilities of each entity are presented in a table.

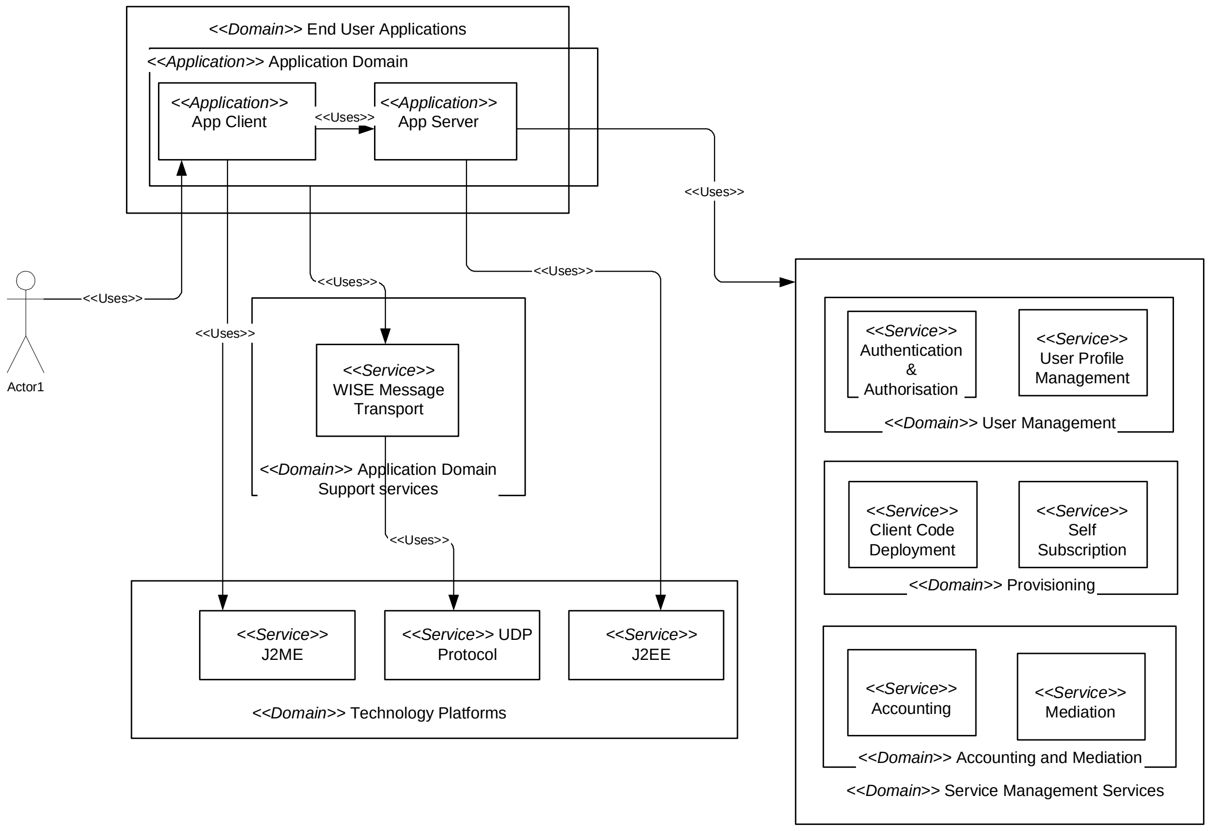

<<*Domain*>> End User Applications
<<*Application*>> Application Domain
<<*Application*>> App Client
<<Uses>>
<<*Application*>> App Server
<<Uses>>
Actor1
<<Uses>>
<<Uses>>
<<Uses>>
<<Uses>>
<<*Service*>> WISE Message Transport
<<*Domain*>> Application Domain Support services
<<Uses>>
<<*Service*>> J2ME
<<*Service*>> UDP Protocol
<<*Service*>> J2EE
<<*Domain*>> Technology Platforms
<<*Service*>> Authentication & Authorisation
<<*Service*>> User Profile Management
<<*Domain*>> User Management
<<*Service*>> Client Code Deployment
<<*Service*>> Self Subscription
<<*Domain*>> Provisioning
<<*Service*>> Accounting
<<*Service*>> Mediation
<<*Domain*>> Accounting and Mediation
<<*Domain*>> Service Management Services

Figure 6. Conceptual structure

The responsibilities of the elements presented in the in conceptual structure are the following:

Conceptual Element	**Responsibilities**
App Client	Provides graphical user interface and handles the user visible subset of data.
App Server	Handles the data status and synchronizes the state between different users.
Service Management Services	Provides a set of common management services as described in D4B. Used services belong to the User Management Domain (Authentication and Authorisation and User Profile Management), Provisioning Domain (Client Code Deployment and Self-Subscription) and the Billing Domain (Accounting and Mediation).
WISE Message Transport	Provides a generic message-based communication service, supporting both synchronous and asynchronous modes. It is based on UDP Service.
J2ME	Java 2 Micro Edition. It is the version for mobile devices of Java.
UDP protocol	The well-known unreliable message service over IP
J2EE	Java 2 Enterprise Edition. It is a java based platform which provides a full set of services for developing and running java server-side application.

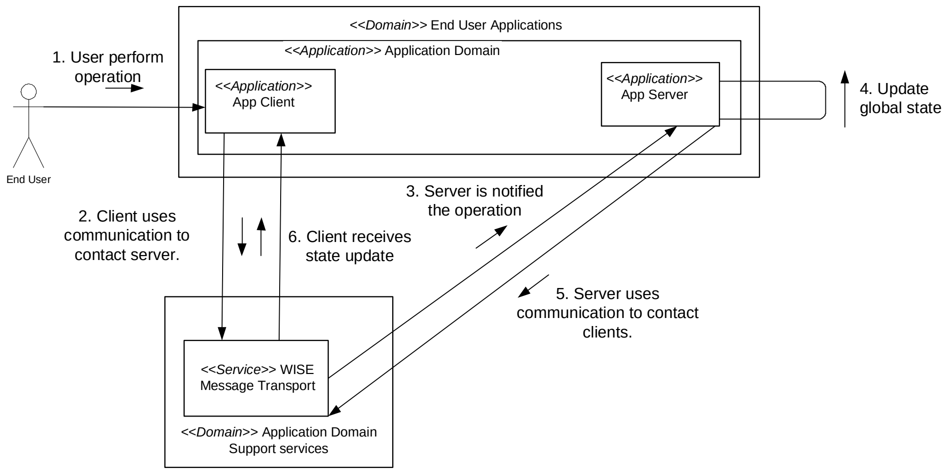
The application is divided into client and server; this implies the choice of client server architectural style. The reasons to select this style are:

(1) users access the service using mobile devices, with limited processing power and memory and therefore, it is obvious to concentrate a computational intensive common, shared part on the server.
(2) a capability to manage several wireless network connections at the same time, results in the server to be a robust, thick server, whereas terminals are clients only hosting the user applications
(3) cient-server style easily achieves scalability.

Entity *Wise Message Transport* is particularly important because, for satisfying the requirements of a real-time service (i.e. the synchronization between the client and the server), a reliable message based protocol is needed (currently many J2ME/J2EE application based uses HTTP to communicate which was proved not adequate for the scope of the project).

3.2.1.2 Behavioural View

Figure 7 presents a collaboration diagram describing the typical scenario of a rich interaction architecture. Typically the users perform operations upon their client applications; such operations are notified to the server through the WISE message transport and modify the state of the system. Once the system state undergoes a change, it has to be communicated to the clients allowing all of them to share a common consistent view of the system.



<<*Domain*>> End User Applications
<<*Application*>> Application Domain
1. User perform operation
<<*Application*>> App Client
<<*Application*>> App Server
4. Update global state
End User
3. Server is notified the operation
2. Client uses communication to contact server.
6. Client receives state update
5. Server uses communication to contact clients.
<<*Service*>> WISE Message Transport
<<*Domain*>> Application Domain Support services

Figure 7. User login collaboration

3.2.1.3 Deployment View

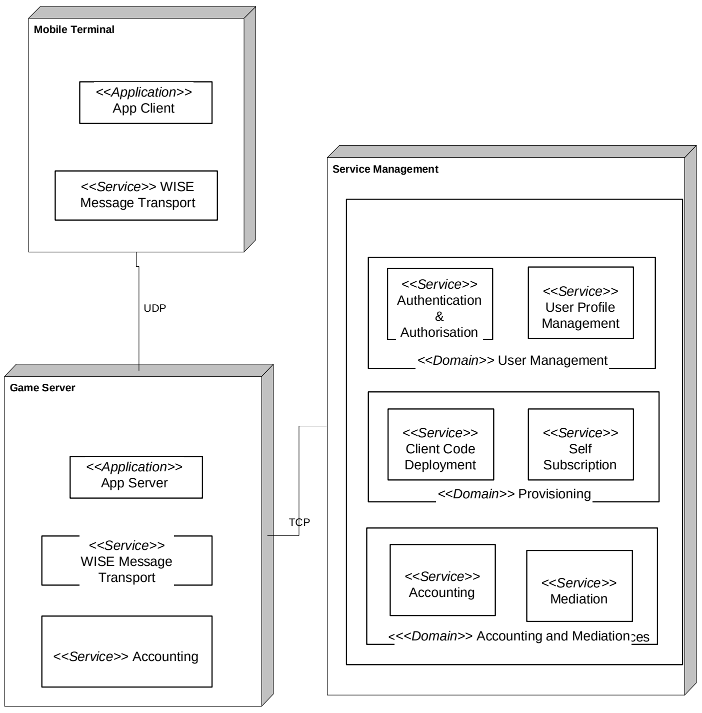

Mobile Terminal
<<Application>> App Client
<<Service>> WISE Message Transport
UDP
Game Server
<<Application>> App Server
<<Service>> WISE Message Transport
<<Service>> Accounting
TCP
Service Management
<<Service>> Authentication & Authorisation
<<Service>> User Profile Management
<<Domain>> User Management
<<Service>> Client Code Deployment
<<Service>> Self Subscription
<<Domain>> Provisioning
<<Service>> Accounting
<<Service>> Mediation
<<<Domain>> Accounting and Mediationes

Figure 8. Conceptual deployment.

It is assumed that a server node handles the management of a number of users with mobile devices. The server handles all synchronization and communication; direct communication between mobile terminals is not allowed. Finally, the management services are most likely in a separate node.
Accounting Service is present on the server to collect usage data and billing queries.

3.2.1.4 Development View

We focus on a typical business model. We skip the topology model because it is too project-specific.

3.2.1.4.1 Business Model

The Business Model instantiated for this typical architecture is depicted in Figure 9, in which only relevant Business Roles and Business Relationships are represented.

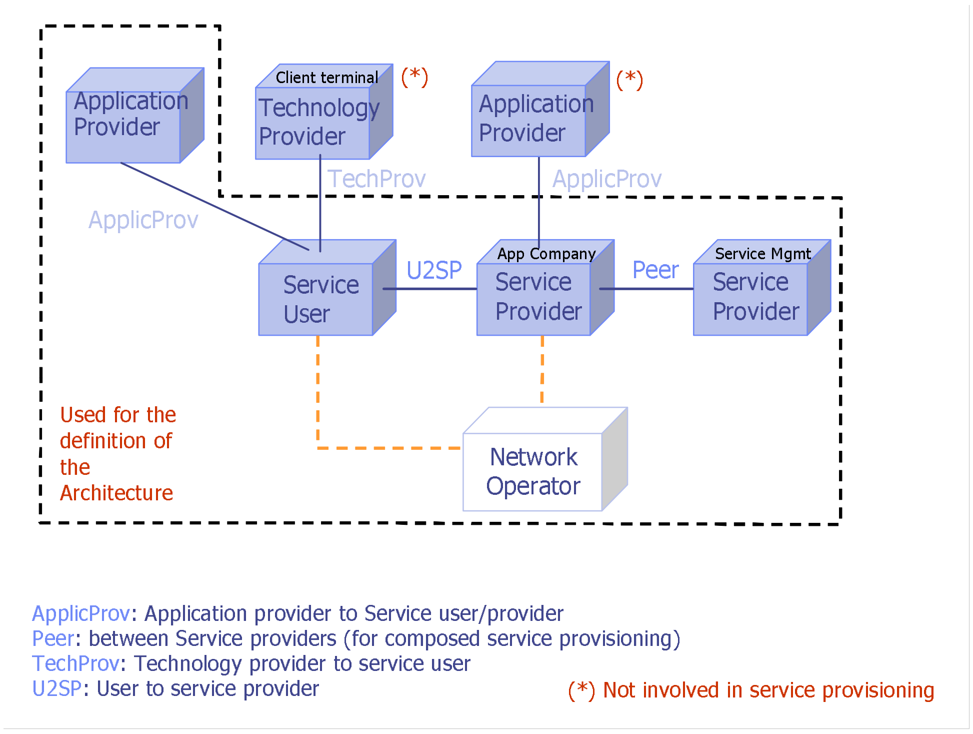

Application Provider
Client terminal
Technology Provider
(*)
Application Provider
(*)
TechProv
ApplicProv
ApplicProv
Service User
U2SP
App Company
Service Provider
Peer
Service Mgmt
Service Provider
Used for the definition of the Architecture
Network Operator
ApplicProv: Application provider to Service user/provider
Peer: between Service providers (for composed service provisioning)
TechProv: Technology provider to service user
U2SP: User to service provider
(*) Not involved in service provisioning

Figure 9. The Business Model

Business roles in dark play some task in the operation of the service. This task can either involve service provisioning (see those roles inside the dashed box) if there will be some software components deployed in a networked structure, or not involve service provisioning (see roles outside the dashed box) if they have a business relationship prior to service provisioning (e.g. Technology provider).

The relationship ApplicProv deserves more attention. This business relationship models download prior to provisioning. The download supports the acquisition from the user side, of the application (i.e. client components) need to access the service. Download can be in principle carried out from both a fixed node (e.g. using any Internet browser) and a mobile node.

3.2.2 Concrete Architecture

The following describes the concrete pilot architecture from different perspectives (or architectural views) according to D4 viewpoints.

3.2.2.1 Structural View

3.2.2.1.1 Inter Component Diagrams

The Inter-component Diagram is depicted in Figure 10. It provides the system level structural view of distributed components and their interconnections.

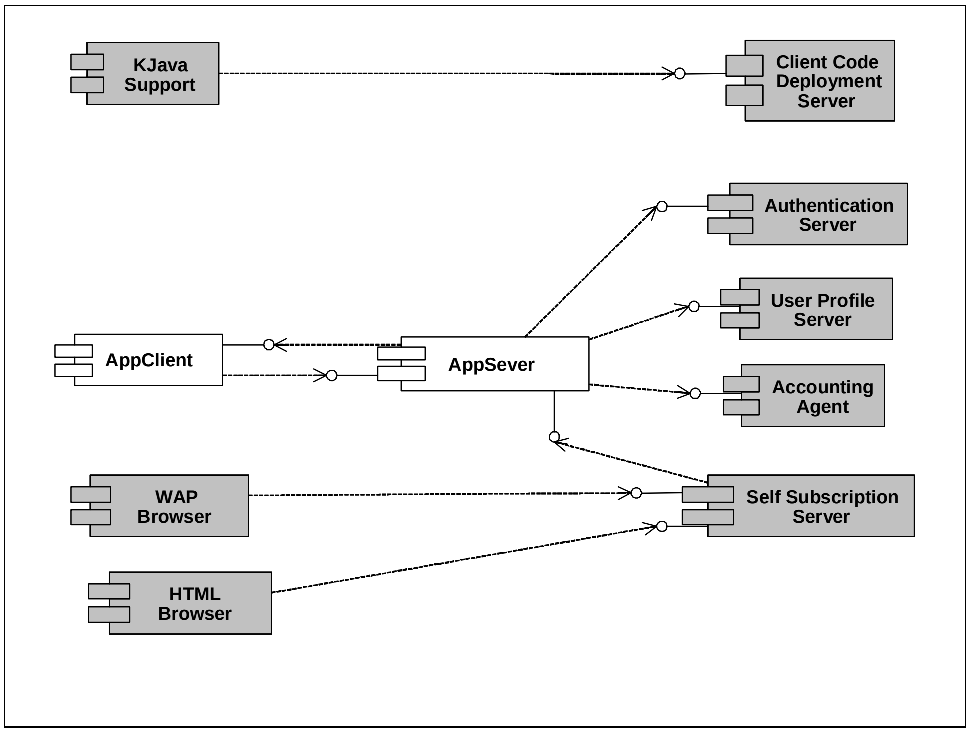


Figure 10: Inter-component Diagram

Service Management Components provides the required services. The Self Subscription Server allows the customer to subscribe a service (in this case the Game Service) by means of a WAP browser available on the mobile phone or an HTML browser from a PC.

Kjava support component is part of the technology already available on the terminal platform; it allows to download the application from a set of sources, i.e. remote server, PC, etc.

In the domain of the Service Provider, two components implement service-specific functionality: component *Game Server* implements the application and the coordination of each service session.
Client Code Download Server is a service-common component, providing support for the end-user to choose and download the application (on the client side) implementing the GUI of the application.

At last, on the client side there is the component providing service-specific functionality (on the left side of Figure 10): the *Game Client* implements the GUI and the processing of data related to an on-going service provisioning. It locally interacts with component *Communication Manager* that supports distributed communication with the remote Service Provider. The component implements the game, and is downloaded from a *Client Code Download Server* as modelled by dependency arrows from the components to the *Client Code Download Server*.

4. PATTERN CATALOG

Here we present a catalog of patterns that can be used to develop wireless services.

4.1 DESCRIPTION OF SELECTED ARCHITECTURAL STYLES AND PATTERNS

This section introduces a set of architectural styles and patterns applicable in wireless service engineering. Table 4 lists a set of architectural styles and patterns and reasons why they are considered to be relevant in the development of service architectures of wireless services. The purpose of the selected styles and patterns is to assist the architect of an end-user service to select appropriate patterns based on the quality requirements set to the new service.

Structural aspects are generally more important than functional aspects when specifying a conceptual architecture for a system. N-Tier Client-Server is the most used style in the conceptual architecture of wireless systems. The Peer-to-Peer style is in the second place. If several pattern alternatives have to be applied for the conceptual architecture, then begin with the pattern that addresses the most important architectural aspect. The tiered style is often used in the conceptual deployment viewpoint.

Table 4. Architectural styles and patterns.

Style or pattern	**Rationale of selection to wireless service engineering**
N-Tier Client-Server	Supports to decompose software functionality into tiers that communicate in the client-server fashion.
Peer-to-Peer	Supports loose-coupling, independence of services without centralized servers.
Blackboard	Is a data centered style that provides flexibility required for adaptation of services.
Pipes & Filters	Is a data flow centered style that supports modifiability and reuse.
Tiered	Is used to partition a wireless system into logically separated tiers. Each tier has a unique responsibility in the system.
Broker	Provides support for distribution transparency.
Layered style	Supports to decompose the software into strict ordered horizontal layers where each layer provides its higher-level layer or layers with a cohesive set of services with a public interface.
Model-View-Controller	Supports separation of concerns and decomposition of responsibilities of application logic from user interfaces.
Presentation-Abstraction-Control	Provides higher independence of components of a service. Generalized from the MVC pattern.

The introduction of each style and pattern is described by the following structure:

- *An overview* gives a brief description of the style or pattern.
- *Intent* describes the situation when the use of the pattern or style is appropriate.
- Application of a style or pattern is described by *conceptual or concrete structures*.
- *Consequences* (benefits and shortcomings) the style or pattern provides.
- *Known uses* give examples of the use of the style or pattern.
- *See also* gives references to patterns that solve similar problems.
- *Variants* gives references to variants of the style or pattern

4.1.1 N-Tier Client-Server (C/S) style

4.1.1.1 Overview

The N-tier Client-Server architecture means an architectural style in which software functionality is decomposed into tiers that communicate in the client-server fashion. The style is a combination of the Tiered style [26] (a specialization of the decomposition style in the module category) and the Client-Server style [26] in the runtime structure category.

Table 5. Summary of the N-Tier Client-Server style

Elements	- Component types: • Clients request services of server components • Servers provide services to client components • Middle-tier components establish communication channels between clients and servers - Environmental elements: network nodes - Connector types: remote procedure calls, the asymmetric invocation of server's services by a client
Relations	Attachment relation - associates clients with the request role of the connector and servers with the reply role of the connector and determines which services can be requested by which clients Allocated-to relation - either static or dynamic allocation of clients and servers to environmental elements
Computational model	Clients request services from servers and wait for the results of those requests
Properties of elements	Client - Name: should suggest the functionality of the component - Type: defines general functionality, the number and types of ports, and required properties - Required hardware properties - Other properties: depend on the type of the component, including quality attributes such as performance and reliability. Server - Name: should suggest the functionality of the component - Type: defines general functionality, the number and types of ports, and required properties - Required hardware properties - The numbers and types of clients that can be attached - Other properties: depend on the type of the component, including quality attributes such as performance (transactions per second) and reliability. Middle-tier - Name: should suggest the functionality of the component - Type: defines general functionality, the number and types of ports, and required properties - Required hardware properties - The number and types of clients that can be attached

- The number and types of servers that can be registered
- Other properties: depend on the type of the component, including quality attributes such as performance and reliability.

Environmental element
- Required hardware properties such as processing, memory and capacity requirements, and fault tolerance

Connector
- Name: should suggest the nature of interactions
- Type: defines the nature of interaction (remote procedure call), the number and types of roles, and required properties
- Other properties: depend on the type of the connector, may include interaction protocols and quality attributes such as performance and reliability.

Topology	N-tiered topology: a node configuration is bound with a division of software functionality into tiers that communicate in the client-server fashion

4.1.1.2 Intent

The Clienet-Server style decouples client applications from the services they use [26]. Its goal is to achieve modifiability and portability [1]. In N-tier Client-Server architectures, client and server components can be independently assigned to tiers or moved from platform to platform, thereby enhancing performance scalability, flexibility, failure recovery, functionality and reliability [8],[31].

4.1.1.3 Conceptual structure

N-tier Client-Server structures are very often presented in architectures that combine the N-tier Client Server style and the deployment style. Figure 11 shows a two-tier Client-Server architecture [26] combined with the deployment style. In this architecture client nodes are usually user interface systems or terminals such as PCs, PDAs or mobile phones on which users run applications. Clients rely on servers for resources, such as files, devices, processing power and application and management software services. Server nodes are usually powerful computers or processes dedicated to managing disk drives (file servers), printers (print servers), network traffic (network servers), or application services. The two-tier Client-Server style has several benefits such as simplicity and efficiency in small systems. The style has some weaknesses such as scalability problems when the variety and number of clients and requests as well the size of the objects increase.

In the three-tier Client-Server architecture, a middle tier has been added between the client environment and the management server environment (Figure 12). The Client-Dispatcher-Server pattern [7] describes one variant of this architecture. This architecture provides a further separation of concerns in the overall architecture. There are a variety of ways of implementing this middle tier, such as transaction processing monitors, messaging servers and application servers. The middle tier can perform queuing, application execution, business rules execution, and database staging. The three-tier architecture has several benefits in comparison with the two-tier architecture such as exchangeability, location and migration transparency and re-configuration of servers [7].

A large variety of servers, clients and middle-tier components may coexist in the three-tier Client-Server architecture. The client nodes may be thin (e.g. simple Web clients), rich (e.g. Web clients with Java applets or ActiveX controls), or fat (e.g. distributed object clients). The three-tier architecture imposes some liabilities such as lower efficiency through indirection and explicit connection establishment as well as sensitivity to change in the interfaces of the middle-tier component [7].

There may be more tiers in a Client-Server architecture, for example, the J2EE platform [12] is a five-tier Client-Server architecture including the following five tiers: Client Tier, Presentation Tier, Business Tier, Integration Tier, and Resource Tier. Two or more of the tiers such as the Presentation Tier, Business Tier, Integration Tier may be allocated to one node in a distributed system [58].

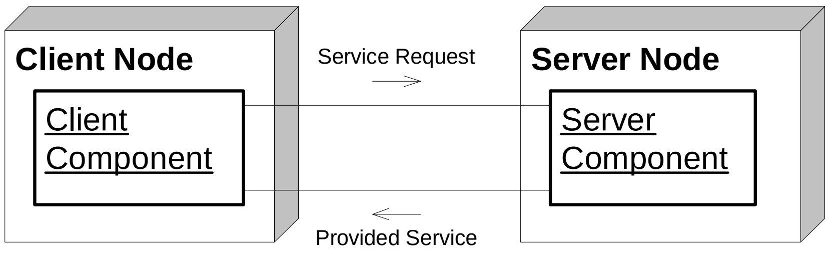


Figure 11. Two-tier Client Server architecture.

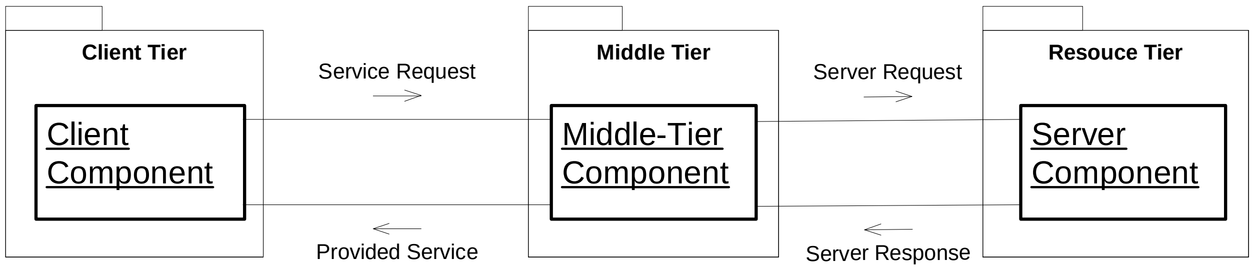


Figure 12. Three-tier Client Server architecture.

4.1.1.4 Known uses

N-tier client-server architectures are used in various systems throughout military and industry. The N-tier client-server style (N>2) has also been applied in many wireless systems, for example to build mobile database applications using the Java technology [58].

The three-tier client-server architecture in Figure 13 has been re-designed from [31]. The middle tier is implemented by the Component Adapter using the Adapter design pattern [16]. It has been allocated to the Server node in this example. Middleware creates instances of a server component as required, and controls the lifetime of the component and its adapter. The component adapter intercepts all service requests between the client component and the server component and applies server services such as transactions and security. The client is unaware of the interception. The server component can use the component adapter for example for querying for client security credentials. [31]

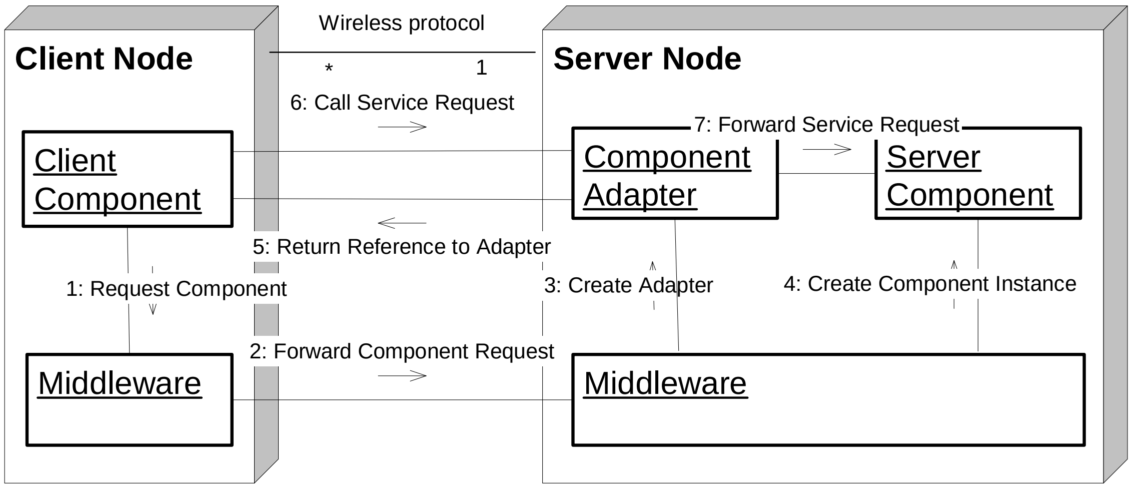


Figure 13. Middleware controls the lifetime of the component and its adapter in the three-tier Client Server architecture (re-designed from [31])

4.1.2 Peer-to-Peer (P2P)

4.1.2.1 Overview

P2P means network architecture, where information is divided between the participating nodes without centralizing it to one server [41]). In P2P model, resources can also be switched between the systems [23]. Because accessing the decentralized resources means operating in an environment of unstable connectivity and unpredictable IP addresses, P2P nodes must operate outside the DNS system and have significant or total autonomy from central servers. [11].

P2P architectures can be classified according to its topology into the pure P2P style. The hybrid P2P style and the mixed P2P style introduced next [40].

4.1.2.2 Intent

P2P is used in distributed computing applications and its aim is to provide maximum flexibility.

4.1.2.3 Conceptual structure

See different alternatives defined later on.

4.1.2.4 Consequences

P2P has the following advantages:

- The user can use resources and data from the other users and receive information from them.
- The amount of servers in the network is directly comparable with the amount of users.
- The work can be divided among the participating users and
- The users can directly communicate with each other [25]

The current P2P systems has the following weaknesses:

- The identities of the used resources and services are not enough transparent and they have to be localized and controlled manually.
- Search of information is limited.
- Lack of co-operation mechanism between the PCs.

Because of the weaknesses in the current P2P systems more advanced P2P architectures based on agents have been developed. When developing P2P architectures it has to be taken account the quality and speed of the information transmitted in the network and its possible misuse.

In Table 10 it is presented the features of P2P architectures and their evolution in time from the hybrid P2P architecture to more advanced models. Alternative P2P architectures have been described more thoroughly later on.

Table 6. Evolution of P2P styles.

Feature ---------------- Time	**Transmitted file-format**	**Reliability**	**Speed**	**Exactness of the information**	**Extensiveness of the searches**	**Privacy**	**Safety**
Pure P2P (Gnutella, Freenet)	All file-formats	Very reliable	High	Quite exact	Extensive, because there is not limit for the searches	High	Middle
Hybrid P2P (Napster)	MP3-files	Unreliable (if the server fails- the whole system fails)	Low	Sometimes very inaccurate, because do not support searches of the sub-strings	Not extensive (every server maintains an index that includes just the files of the customers attached to it)	Low	Middle
Advanced P2P	All file-formats	Reliable	High	Exact	Exhaustive	High	Middle

Table 7. Evolution of P2P styles continues.

Feature ---------------- Time	**Nr of users**	**User-friendliness**	**Focusing of the inf.**	**Effectiveness of the resource use**	**Dependency on server**	**Operating System**	**Total effectiveness**
Pure P2P (Gnutella, Freenet)	Low	Low	Low	Low	Server can be chosen freely.	Almost all platforms	Low
Hybrid P2P (Napster)	High	High	High	High	Server can not be chosen	Windows or Mac	High
Advanced P2P	High	High	High	High	Server can be chosen	Almost all platforms	Highest

The comparison made in Table 6 proves that both the pure P2P model and the hybrid P2P model have disadvantages and advantages the removal of which has at some extent succeeded in the more advanced P2P models.

4.1.2.5 Known uses

The P2P architectural style has been used in Gnutella and Freenet.

4.1.2.6 See also

The architecture of a Peer implements the Layered Architectural Pattern. The functions are layered one on top of the other. The Peer-to-Peer system uses Pipes and Filters Pattern to effectively transfer data from one system to another. The Broker pattern is used in Peer-to-Peer systems. Peers access other peers or the server through the Broker Pattern. It acts as an interface between the system and the user.

4.1.2.7 Variants

4.1.2.7.1 Pure P2P

4.1.2.7.1.1 Overview

Nodes of the pure P2P are peers that can act as clients (like mobile phones) and servers. A peer has the same capability as its neighbors without a centralized router. The pure P2P has two routing structures and all nodes of the network are equal. The first one is a distributed catalogue and the other direct messaging.

4.1.2.7.1.2 Intent

Pure P2P can be used in small wireless systems to search for information in an extensive way and deliver data in all file formats. Pure P2P suits well to dividing information between limited number of users.

4.1.2.7.1.3 Conceptual structure

Figure 6 presents the conceptual structure of the Gnutella that is a pure P2P architecture.

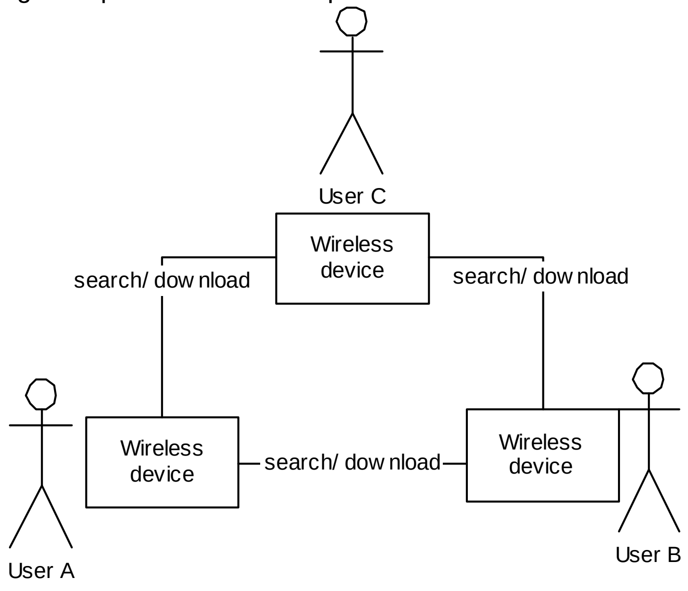


Figure 14. Gnutella as an example of the pure P2P style.

The functionality of Gnutella is described in the following [21]:

1) When you start up Gnutella your "servant" (mobile device) knows nothing of anything that has to do with gnutellaNet. This is why you have to "add" an IP to connect to. Once your servant does connect to another servant, then they start exchanging information. Things like "how many other servants are there and what are their IPs" are exchanged over this link. Your servant may connect to many other servants; it will communicate with all of them. Gnutella's architecture consists of dynamically changing amount of

nodes that uses TCP/IP protocol. As the connection has been achieved, the node use HTTP (Hypertext Transfer Protocol) protocol to communicate the communication happens via PING massages.

2) When you want to find a file, you type in the name of the searched file. Your servant then sends that to all of the servants it is connected to, and all of them send it to all the servants they are connected to, and so on. But, each servant also searches the files it knows it is sharing and sends those results back to you.

3) Then you look through the list of files and decide to download the one you want. When you start the download, your servant tries to connect with the servant that reported the match; if it can connect it will start the download. If it cannot connect, usually because of a firewall, it will instead send a download request to all the servants it is connected to. The request will then travel the same way as the original search and eventually get to the servant that reported the match and it will try to connect back to you. This is how it is possible for Gnutella to work if one of the hosts is behind a firewall.

4.1.2.7.1.4 Consequences

The advantages and disadvantages of P2P style are presented in Table 6.

4.1.2.7.1.5 Known uses

Example implementations are Gnutella [21] and Freenet (http://freenet.sourceforge.net/).

4.1.2.7.2 HYBRID P2P

4.1.2.7.2.1 Overview

In hybrid P2P the central server is responsible for maintaining a registry of shared information and responding to queries for that information. The peers like mobile phones or PDAs are responsible for hosting the information, communicating what is to be shared to the central server, and downloading it to other peers upon request. This is centralized but not in the conventional Client Server sense. Route terminals are used to hold catalogues of addresses. They are referenced by a set of indexes that determine an address set.

4.1.2.7.2.2 Intent

Hybrid P2P suits exchanging information between a huge number of wireless users by holding a central registry about all the information.

4.1.2.7.2.3 Conceptual structure

Figure 15 presents the conceptual structure of Napster that is a hybrid P2P style.

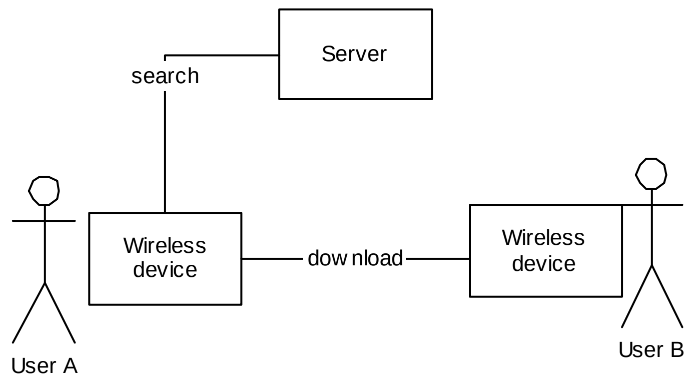


Figure 15. Napster as an example of the hybrid P2P.

Napster functions in the following way: [21]

1) When you start up Napster your "client" that means the application that runs on your wireless device connects to a predefined "server". Your client identifies it to the server and sends information about yourself to it; most importantly it sends a list of the files you are sharing. The server then keeps track of all the clients that are connected to it and which files they are sharing. Napster uses FTP (File Transfer Protocol) as a communication protocol.

2) When you want to find a file, you type in what you are searching for. Your application sends your request to the server and the server then sends back a list of all the clients that are sharing files that match your request. This list is then displayed to you.

3) You look through the list and decide the one you want to download. When you select it and start the download your client knows which client is sharing this file and connects directly to it, without the server's intervention.
   Consequences

The advantages and disadvantages of the hybrid style are presented in Table 6.

4.1.2.7.2.4 Known uses

Hybrid architectural style has been used in Napster for transmitting music files.

4.1.2.7.3 Agent-based P2P architecture

4.1.2.7.3.1 Overview

In an agent-based P2P architecture the user communicates with agents that are located inside the wireless device and the agents can work on behalf of the human-users. The agents can learn from the past, work together and consult each other. The agent-based P2P architectures also fulfill the deficiencies of the current P2P architectures by offering most of the functions needed in ideal P2P systems. [25].

4.1.2.7.3.2 Intent

Agent based P2P architecture can be used for transmitting effectively all data formats in various wireless applications.

4.1.2.7.3.3 Conceptual structure

Figure 16 presents the conceptual structure of the agent-based P2Pstyle.

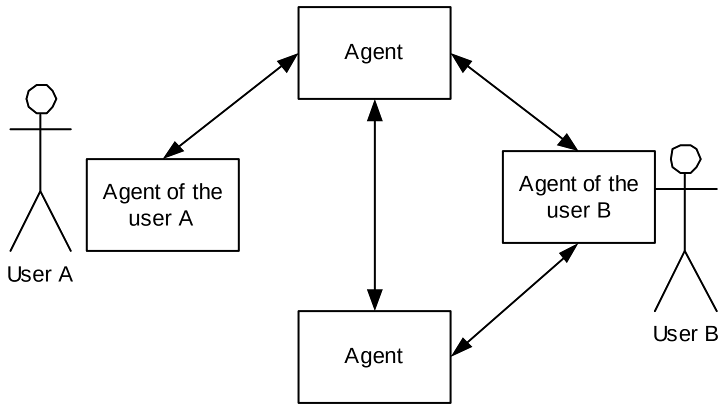


Figure 16. Agent-based P2P architecture style.

4.1.2.7.3.4 Consequences

The advantages and disadvantages of the agent based P2P-style is presented in Table 6.

4.1.2.7.3.5 Known uses

The agent-based P2P style has been used so far in pilot applications.

4.1.3 Blackboard

4.1.3.1 Overview

In Blackboard several specialized subsystems assemble their knowledge to build a possibly partial or approximate solution. The idea behind the Blackboard architecture is a collection of independent programs that work co-operatively on a common data structure. Each program is specialized in solving a particular part of the overall task. The specialized programs are independent of each other. The direction taken by the system is determined by the current state of the progress.

4.1.3.2 Intent

The Blackboard architectural pattern is useful for problems for which no deterministic solution strategies are known. Blackboard suits best to the systems, where it is needed scalability in the form of adding consumers of data without changing the procedures and modifiability in the form of changing who produces and consumes which data. [4]

4.1.3.3 Conceptual structure

In Figure 17 it is presented the Blackboard architectural pattern, where all application components located in a wireless device have access to a shared data space or blackboard. In the Blackboard pattern application components look for particular kinds of data objects on the blackboard and produce new data objects that are added to the blackboard. An optional co-ordinate component can be used to coordinate the activation of application components. [15]

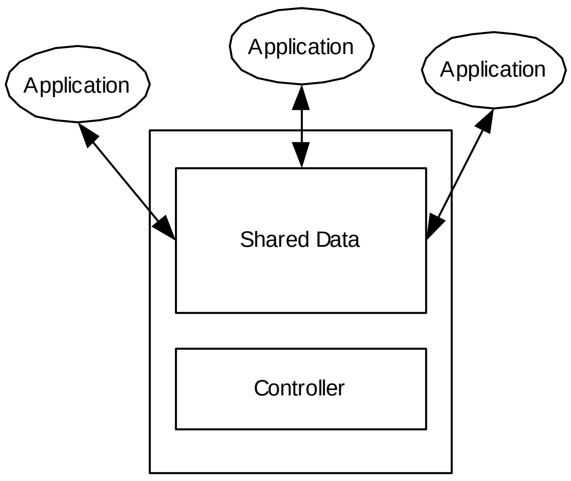


Figure 17. Blackboard architectural pattern.

4.1.3.4 Consequences

The advantages of the Blackboard pattern are:

- Easy to maintain by adding data-types.
- Experimentation with different algorithms is possible.
- Support for changeability and maintainability.
- Reusable knowledge sources.
- Support for fault tolerance and robustness.

The Blackboard pattern has some liabilities:

- Performance is generally rather low
- Hard to identify unfulfilled responsibilities
- Difficulty of testing.
- No good solution is guaranteed.
- Difficult of establishing a good control strategy.
- Low efficiency. Blackboard systems suffer from computational overheads in rejecting wrong hypothesis.
- High development effort.
- No support for parallelism. This means that concurrent access to the central data on the blackboard must be synchronized.

4.1.3.5 Known uses

The first Blackboard system was the HEARSAY-II speech recognition system from the early 1970's. It was developed as a natural language interface to a literature database. Blackboard has also been used in the HASP system that was designed to detect enemy submarines. In this system, hydrophone arrays monitor a sea area by collecting sonar signals.

4.1.3.6 See also

The Blackboard pattern has no similatities with other patterns.

4.1.4 Pipes and Filters

4.1.4.1 Overview

The Pipes-and-Filters style emphasizes the incremental transformation of data by successive components. Filters are stream tansducers that incrementally transform data, use little contextual information, and retain no state information between instantiations. Pipes are stateless and simply exist to move data between filters. A pipe has a source end that can only be connected to a filter's output port and a sink end that can only be connected to a filter's input port.

4.1.4.2 Intent

The Pipes-and-Filters style intents to view the system as a series of transformations on successive pieces of input data. Data enters the system and flows through the components one at a time until they are assigned to some final destination, output or a data store. Thus, the Pipes-and-Filters style intents to achieve simplicity, maintainability and reusability of a system.

4.1.4.3 Conceptual structure

Figure 18 presents the conceptual structure of the Pipes-and-Filters style.

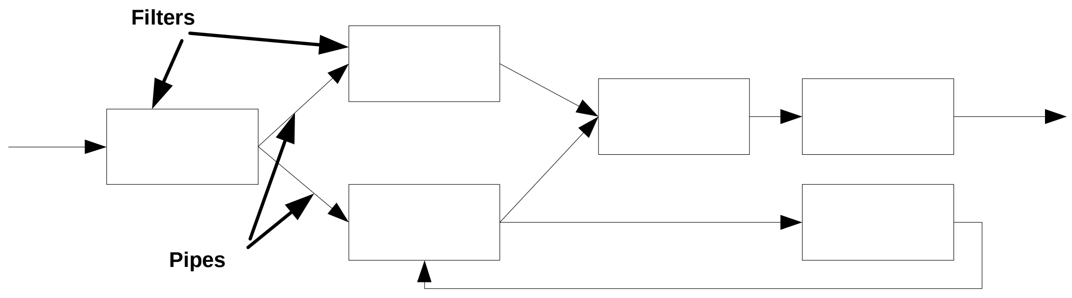


Figure 18. The Pipes-and-Filters style.

4.1.4.4 Consequences

The Pipes-and-Filters style has the following advantages:

- Simplicity due to limited ways to interact with the environment.
- Simplicity is achieved by a composition of primitive functions.
- Filters are reusable black-box components.
- Hierarchically composable; any combination of filters connected by pipes, can be packaged and used in the external world as a filter.
- Parallel and distributed systems can easily be made; it enhances performance without modifications

The Pipes-and-Filters has also some liabilities:

- Batch mentality is implicitly encouraged; interactive applications are difficult to create.
- Filter ordering is difficult; there is no way for filters to cooperatively interact, more often control in embedded into data to be transformed.
- Performance might be poor due to several reasons: 1) Input needs to be transformed into tokens and thus, every filter pays this parsing/unparsing overhead. 2) The filter may need a buffer of unlimited size. Otherwise the system could deadlock. 3) Each filter operates as a separate process and some overhead follows from each time it is invoked.

4.1.4.5 Known uses

- UNIX family of operating systems

- DSP software (at the physical layer)
- The architecture of a system where sales agents submit orders vie handheld devices wirelessly to an office-based server that co-ordinates the orders [24].

4.1.4.6 See also

The other data-flow-centered style used in classical data processing systems is the batch sequential style. The style processes steps or components that are independent programs based on the assumption that each step has to be completed before the next step starts. Each batch of data is transmitted as a whole between steps.

4.1.5 Tiered style

4.1.5.1 Overview

The Tiered style is used to partition a wireless system into logically separated tiers. Each tier has a unique responsibility in the system. A tier is logically separated from other tiers in the system, and is loosely coupled with adjacent tiers.

4.1.5.2 Intent

The responsibility of conceptual elements in a wireless system, data flow among elements, locality of processing, the presence and use of communication channels, and allocation to conceptual nodes of the system all tend to be presented using the Tiered style.

4.1.5.3 Conceptual structure

An example of the conceptual structure of the Tiered style is shown in Figure 19 [12]. The client tier is responsible for user interaction, user interface presentation and client devices. The presentation tier is responsible for single sign-on, session management, content creation, format and delivery. The business tier is responsible for business logic, transactions and data services. The integration tier is responsible for resource adapters, legacy, external systems, rule engines and workflows. The resource tier is responsible for resources, data and external services. The use relations between the tiers are allowed to be bi-directional or symmetric.

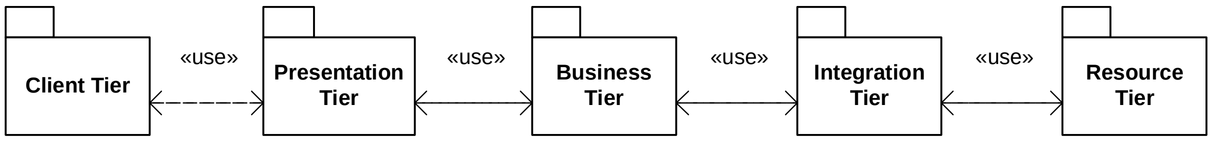


Figure 19. An example of the Tiered style (re-designed from [12])

4.1.5.4 Known uses

- The J2EE platform is a multitiered system [12]. There are the following five tiers in the tiered model: Client Tier, Presentation Tier, Business Tier, Integration Tier, and Resource Tier.
- The three-tiered style has been used to decompose a mobile service system in the PALIO (Personalised Access to Local Information and services for tOurists) project [3].

4.1.5.5 See also

The Tiered style is often combined with another style (e.g. the Client-Server style) which results in a combined style such as the N-Tier Client-Server style (Section 4.1.1) that shows the allocation of the components of the Client-Server style into conceptual tiers. Tiers are often confused with layers [8].

4.1.6 Broker

4.1.6.1 Overview

By using the Broker pattern, a wireless application can access distributed services by sending massage calls to the appropriate object (like network server) through the broker. The Broker architectural pattern reduces the complexity involved in developing distributed applications because it makes distribution transparent to the service developer [7].

4.1.6.2 Intent

The Broker architectural pattern is applied to structure distributed wireless systems with decoupled components that interact by remote service invocations. A broker component is responsible for coordinating communication between the client, server and proxies, such as forwarding requests, as well as transmitting results and exceptions to the clients like mobile phones. Figure 20 presents the main elements of the Broker architectural pattern [7].

4.1.6.3 Conceptual structure

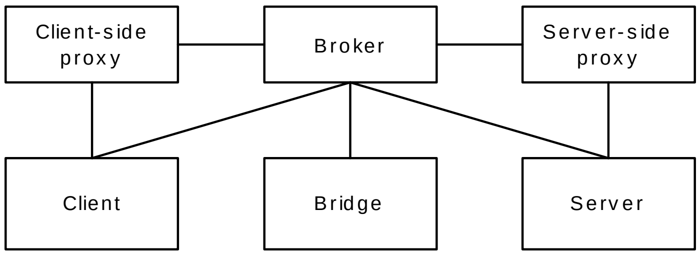


Figure 20. Broker architectural pattern.

4.1.6.4 Consequences

The Broker architectural pattern has the following benefits:

- Portability enhancements: A broker hides OS and network system details from clients and servers by using layers, such as APIs, proxies and adapters.
- Interoperability with other brokers: Different brokers may interoperate through a bridge if they understand a common protocol for exchanging messages.
- Reusability of services: When building new wireless applications, brokers enable the application functionality to reuse existing services
- Location transparency: A broker is responsible for locating servers, so clients like mobile phones need not know where servers are located.
- Changeability and extensibility of components: If the server implementation changes without affecting interfaces clients should not be affected.

The disadvantaged of the Broker architectural pattern are:

- Restricted efficiency: Wireless applications using brokers may be slower than applications written manually.

- Lower fault tolerance: Compared with non-distributed wireless applications, distributed broker systems may incur lower fault tolerance.
- Testing and debugging may be harder: Testing and debugging of distributed wireless systems is tedious because of all the components involved.

4.1.6.5 Known uses

- CORBA
- IBM SOM/DSOM
- Microsoft's OLE
- WWW
- ATM-P

4.1.6.6 See also

In the Proxy pattern the proxy encapsulates the interface and remote address of the server. The Mediator design pattern replaces a web of inter-object connections by a star configuration in which the central mediator component encapsulates collective behavior by defining a common interface for communicating with objects.

4.1.6.7 Variants

The Transceiver-Parcel and Broker as Intermediary patterns [39] help to design an elastic architecture that commits to the Broker idea and can be extended by adding components or reduce by removing them. The Broker as Divorce Attorney pattern [39] helps to group and distribute the components of an application across processes and processors in may different ways. The Broker as Matchmaker pattern [39] can be used in the Broker architecture when coupling is a minor concern, and efficiency the highest priority

4.1.7 Layered style

4.1.7.1 Overview

The layered architectural style (Figure 21) helps to decompose the software into strict ordered horizontal layers where each layer provides its higher-level layer or layers with a cohesive set of services with a public interface [7] [8].

4.1.7.2 Intent

The Layered style suits best to the system where the tasks can be divided to application specific and generic tasks. The generic tasks are specific to the underlying computing platform. In layered style portability across computing platforms is important. [4]

4.1.7.3 Conceptual structure

Figure 5 presents the conceptual structure of the layered style. Layers represent virtual machines. Each virtual machine provides a cohesive set of services with a public interface. The interface should be independent of a particular platform. The layers should be independent of each other as much as possible. The layer structure may remain stable even though the content of the layers may change. The allowed-to-use relation associates layers with each other [8]. The usage of layers generally flows downward. Each layer typically communicates only with the layer immediately below it. Upward usage is regarded as exceptions to the rule and unrestricted upward usage is not allowed.

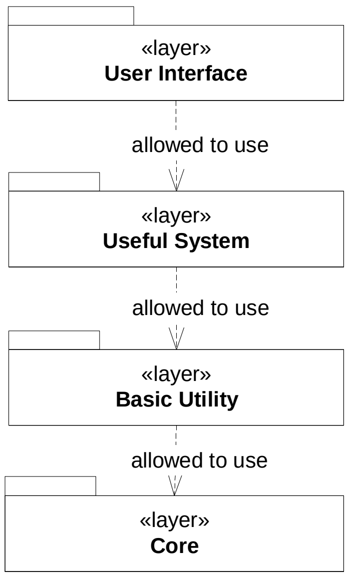


Figure 21. The layered style.

4.1.7.4 Consequences

The benefits of the layered architectural style are:

- Reuse of layers. If an individual layer embodies a well-defined abstraction and has a well-defined and documented interface, the layer can be used in multiple contexts.
- Support for standardization: Clearly defined and commonly accepted levels of abstraction enable the development of standardized tasks and interfaces.
- Dependencies are kept local.
- Exchangeability as individual layer implementations can be replaced by semantically equivalent implementations without too great effort.
- Easy to maintain as there are few dependencies on other layers.
- A wireless system can be built using layers of increasing abstraction.
- Implementation of each layer can be exchanged as long as the protocols are the same.
- Portability: low-level dependencies are hidden within a layer
- Enable some determination of the scope of the changes [8].

The layered architectural style has the following disadvantages:

- Cascades of changing behavior [7]
- Lower efficiency
- Unnecessary work
- Difficulty of establishing the correct granularity of the layers.
- Decreased performance (several switches of method context)
- Identification and delimitation of a layer is often difficult [18]

4.1.7.5 Known uses

- OSI protocol
- Virtual machines
- APIs
- Information systems
- Windows NT

4.1.7.6 See also

The Layered style is often confused with other architectural styles such as the decomposition style and the Tiered style [26]. The styles serve different concerns although their components may have a one-to-one correspondence in an architecture [8].

A Microkernel architecture [7] can be considered as a specialized layered architecture. Overall PAC structure (described later) is a tree of PAC nodes. PAC emphasizes that every logical node consists of three components: presentation, abstraction and control while the Layered style does not prescribe any subdivision of an individual layer [7].

4.1.8 Model-view-controller

4.1.8.1 Overview

The Model-View-Controller architectural pattern (MVC) divides an interactive application into three components. The model contains the core functionality and data. View displays information to the user. Controllers handle user input. Views and controllers together comprise the user interface of the mobile device. A change-propagation mechanism ensures consistency between the user interface and the model.

4.1.8.2 Intent

MVC emphasizes modifiability and portability by applying separation of input and output-devices and use of the unit-operation of the part-whole decomposition [4]. The separation is achieved by dividing an application into three components.

4.1.8.3 Conceptual structure

In Figure 10 it is presented the functionality of MVC architectural pattern [7].

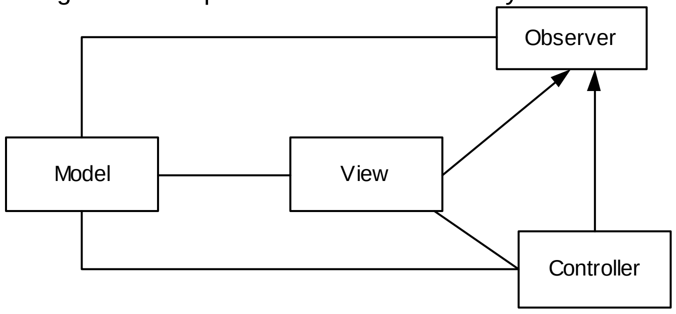


Figure 22. Model-View-Controller architectural pattern.

In the MVC architectural pattern the model component contains the functional core of the application. It encapsulates the appropriate data and exports procedures that perform application-specific processing. The model also provides functions to access its data.

The View components present information to the user by the aid of different views. Each view defines an update procedure and when it is called, a view retrieves the current data values to be displayed from the model and puts them on the screen.

The controller components accept user inputs as events.

4.1.8.4 Consequences

The application of MVC has several benefits:

- Multiple views of the same model

- Synchronized views
- Pluggable views and controllers
- Exchangeability of look and feel. A port of an MVC application to a new platform does not affect the functional core of the application.
- Framework potential

MVC has the following liabilities:
- Increased complexity
- Potential for excessive number of updates
- Intimate connection between view and controller
- Close coupling of views and controllers to a model
- Inefficiency of data access in view
- Inevitability of change to view and controller when porting
- Difficulty of using MVC with modern user-interface tools

4.1.8.5 Known uses

- The best-known example of the use of MVC is the user-interface framework in the Smalltalk environment. MVC was established to build reusable components for the user interface. The tools that make up the Smalltalk development environment share these components. The Document-View variant of the MVC-pattern is integrated in the Visual C++ environment for developing Windows applications.
- The MVC pattern has been used to discover an extensible, maintainable, scalable and portable structure for mobile applications in the PALIO (Personalised Access to Local Information and services for tOurists) project [3].
- The MVC pattern has been used in designing wireless clients with the Java technology [52]

4.1.8.6 See also

The Presentation-Abstraction-Control pattern takes a different approach to decoupling the user-interface aspects of a system from its functional core. Its abstraction component corresponds to the model in MVC, and the view and controller are combined into a presentation component. Communication between abstraction and presentation components is decoupled by the control component. The interaction between presentation and abstraction is not limited to calling an update-procedure.

The Model-View-Controller (MVC) architectural pattern combined with the Facade [16] and Proxy [16] patterns can be used for supporting data models in both online and offline operating modes in wireless systems. The MVC pattern isolates the GUI from the data access, which is done via the network from the server or locally from the device's database. This separation is fundamental for the implementation of a disconnected mode of operation. The Facade structural design pattern can be used to hide the complexity of the client-side data model implementation. The Proxy design pattern can be used to abstract the logic that deals with accessing remote data, such as the client/server communication protocol, and any related optimization, such as caching. The proxy component of the pattern provides a placeholder to another object to control access to that object.

4.1.9 Presentation-abstraction-control (PAC)

4.1.9.1 Overview

The Presentation-Abstraction-Control architectural pattern (PAC) defines a structure for interactive wireless systems in the form of a hierarchy of cooperating agents.

4.1.9.2 Intent

The Presentation-Abstraction-Control architectural pattern defines a structure for interactive wireless systems in the form of a hierarchy of co-operating agents [7]. Every agent is responsible for a specific aspect of the application's functionality and consists of three components: presentation, abstraction, and control. This subdivision separates the human-wireless device interaction aspects of the agent from its functional core and its communication with other agents. PAC supports modifiability and scalability [4].

4.1.9.3 Conceptual structure

In Figure 11 it is presented the internal structure of a PAC agent.

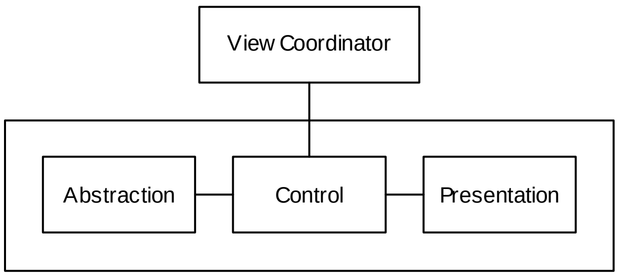


Figure 23. Internal structure of a PAC agent.

The presentation of an agent provides the visible behavior of the PAC agent. Its abstraction component maintains the data model that underlies the agent and provides functionality that operates on this data. Its control component connects the presentation and abstraction components and provides functionality that allows the agent to communicate with other PAC agents. Consequences

The PAC architectural pattern has several benefits:

- Separation of concerns. Separate agents represent different semantic concepts in the application domain.
- Support for change and extension
- Support for multi-tasking
- When the principle of separation of responsibilities is consequently employed the resulting systems can be easily adjusted and extended
- Agents are reusable and portable
- Agents my be distributed on different host/processors
- Multiple user support possible when synchronized

The disadvantages of the PAC architectural pattern are:

- Increased system complexity
- Complex control component. The individual roles of control components should be strongly separated from each other to guarantee a good quality for the system.
- Efficiency. The overhead in the communication between PAC agents may impact the system efficiency.
- Applicability. The smaller the atomic semantic concepts of an application are, and the greater the similarity of the user interfaces in wireless devices, the less applicable this pattern is.
- Communication and permanent data transfers reduce system performance

4.1.9.4 Known uses

- Network traffic management
- Mobile Robot
- Reconfigurable systems

4.1.9.5 See also

The Model-View-Controller pattern separates the functional core of a software system from information display and user input handling. MVC however defines its controller as an entity responsible for accepting and translating it into internal semantics. This means that MVC divides the user-accessible part into view and control.

4.2 WIRELESS-SPECIFIC PATTERNS

Several issues (see section 1.1) in the context of wireless service engineering are specific of this context. The idea is to collect a set of wireless specific patterns that address such issues. This attempt is similar to that presented in [59].

4.2.1 Reduced Mark-up Language

4.2.1.1 Overview

The problem of presenting complex structured information on a limited device can be solved using an ad-hoc mark-up language such as WML, which requires a small footprint browser.

4.2.1.2 Intent

The reduced mark-up language addresses the problem of representing fairly complex information on a device with limited capabilities. The limitations that are addressed consists essentially

- limited display capabilities,
- limited device resources usable for presentation software,
- missing development effort for the implementation of an ad-hoc presentation software.

Since several recent mobile devices come in bundle with WML browsers, it is possible to leverage this built-in capability to implement all the presentation related features on the client side.

4.2.1.3 Conceptual structure

Adopt a simplified mark-up language such as WML to encode documents
WML is standard on most mobiles, therefore this is the solution of choice

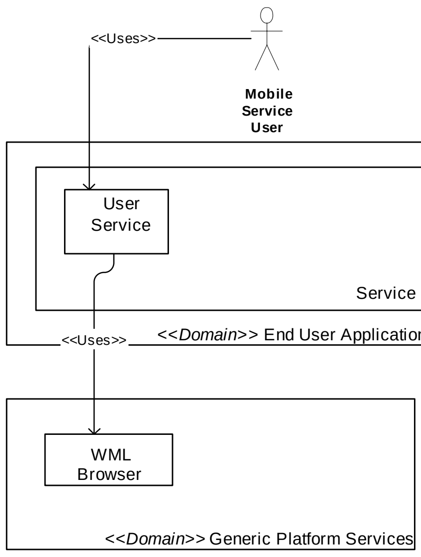

<<Uses>>
Mobile Service User
User Service
Service
<<Uses>>
<<*Domain*>> End User Applications
WML Browser
<<*Domain*>> Generic Platform Services

Figure 24. Conceptual structure of the reduce mark-up language pattern.

4.2.1.4 Consequences

The presentation can be formatted even though to a limited extent.
The browser is fairly standardized and requires reduced power and graphics capabilities.

4.2.1.5 Known uses

Several public-access web portals developed stripped down versions for mobile phones using WML.

4.2.1.6 See also

4.2.2 Connection-less protocols

4.2.2.1 Overview

When a client needs a frequent and low latency notification of events from a server on a wireless network, the use of TCP/IP may not meet the latency requirements.
Solution consists in adopting the UDP/IP protocol and introducing some packet loss detection and recovery mechanism at a higher level.

4.2.2.2 Intent

This pattern has the purpose of limiting the bandwidth occupation of the protocol to obtain low latency.

4.2.2.3 Conceptual structure

To reduce bandwidth, we define an application specific protocol that is based on UDP.
UDP reduces significantly the overhead of establishing a connection with respect to TCP and protocols based on it, such as HTTP.

4.2.2.4 Consequences

The UDP allows a lower overhead and bandwidth occupation, therefore allows notification of events with very low latency.
On the other side UDP does not guarantee the delivery of packets. Therefore the application must be able to handle missing informations.

4.2.2.5 Known uses

WISE Pilot 2, in its first iteration, uses this pattern to cope with the limited bandwidth of GPRS network.

4.2.2.6 See also

4.2.3 Multiple presentations

4.2.3.1 Overview

In a network environment, especially with wireless devices, the characteristics of client applications often are very different. It is very useful to provide the users with a single point of access to a service and automatically adapt to the client/device features.

4.2.3.2 Intent

We need access to the same information through different media and devices. As the user changes device he/she must be offered the same service according to the devices capabilities.
In general the same service must be presented through different channels. But in practice each device has different capabilities and the details of the presentation that can depend heavily on the device or client application capabilities.
This pattern provides a method to automatically detect the type of device/client and switch to the presentation mode that best suits it.

4.2.3.3 Conceptual structure

Add a presentation layer for each specific device and let it select the features that can be show on the device and organize the contents in the most suitable way.
All the versions of the user interface are based on the same back-end component; they only adapt the information in the back-end to specific languages/devices.
In addition we need an entry point that can redirect the user to the most suitable presentation on the base of the user’s device capabilities.

As an example using JSP to provide access to devices using different languages would yield the following structure:

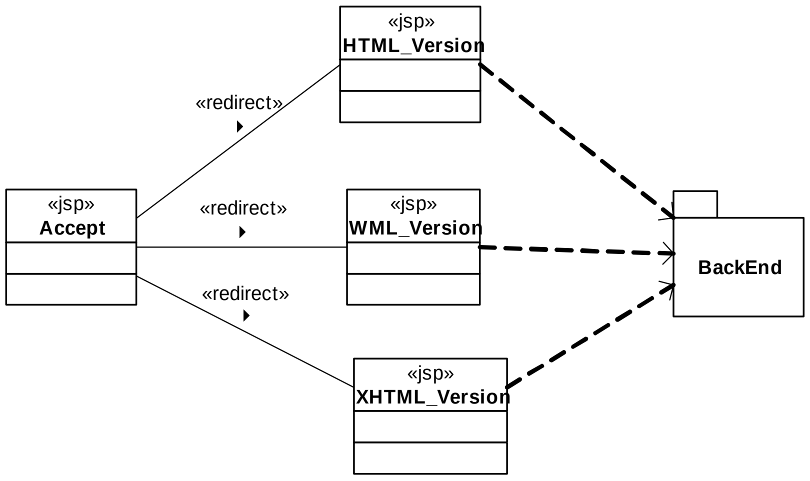


Figure 16.25: Multiple presentations..

The Accept JSP is the target for the connection for all the types of browsers, it identifies the type of browser and redirects it to the version of the presentation most appropriate for that browser.
The presentation versions are thin and focus only on the presentation language and on the level of detail. The back-end component hides all the complexities and provides a unique source of information for all the channels.

4.2.3.4 Consequences

The same information is at the base of all the presentations
Each version of the presentation is tailored for a specific device
Each version can filter the information and provide the suitable level of detail

4.2.3.5 Known uses

This approach is used in several web portals to recognize the browser and use the suitable extension. This works essentially to distinguish between Netscape and Microsoft browsers.
The same approach is used to identify the type of mobile device that connects to a WAP server. This distinction is very important because the support for WML is not uniform across different devices. In this case it is possible to adapt the WML to the devices that connect to the service.

4.2.3.6 See also

It could be used together with the MVC pattern: the multiple presentation serves the purpose of selecting the view most appropriate for the client.

5. WISA BASIC SERVICES

The WISA basic services presented in the following section are structured by the service taxonomy selected to WISA/RA. A basic service belongs to one of the taxonomy domains. End-user services will not be included into WISA basic services. Technology platform services are not described equally to the other basic services but only their selection criteria have been presented.

Because it is impossible to present all basic services in a single architectural model or diagram each basic service is presented according to the documentation pattern described in D4 Part B.

This section introduces a set of basic services suitable for wireless services and known so far. Application support services that now provide only two services for game entertainment will be supplemented in the next iteration phase with supporting services and application frameworks useful in future wireless services, e.g m-health services. Also location based services need further studies.

Although some kinds of solutions for generic platform services have already identified, they need to adapted to heterogeneous environments, e.g. mobile terminals and PDAs, and extend with variability inside the services, and furthermore, validate their usefulness in the wireless services.

5.1 APPLICATION DOMAIN SUPPORT SERVICES

5.1.1 3D Game Engine (X-Forge™ Technology)

Identification:
The X-Forge™ 3D Game Engine is a complete C++ based cross-platform game engine and suite of tools for developing advanced 3D games for major mobile platforms.

Source:
This service is commercially available from Fathammer (http://www.fathammer.com).

Special terms and rules:

Quality attributes:

Table 8. Quality requirements and theirs intended realization.

Requirement	Definition	Realization
Performance	Processing power in the target devices is very limited.	Code is as carefully optimized as possible.
Portability	Game engine must be portable into the several operating environments.	Hardware depended parts of the platform are separated from the generic part with an abstraction layer.

Features:
Graphics Support
- 2D
- 3D
  - Hardware accelerated

Software accelerated

Network Support
- Bluetooth
- WLAN
- 3G

Operating System
- Symbian OS
- Microsoft Smartphone
- Microsoft Pocket PC
- Palm OS
- Mobile Linux

Conceptual structure:

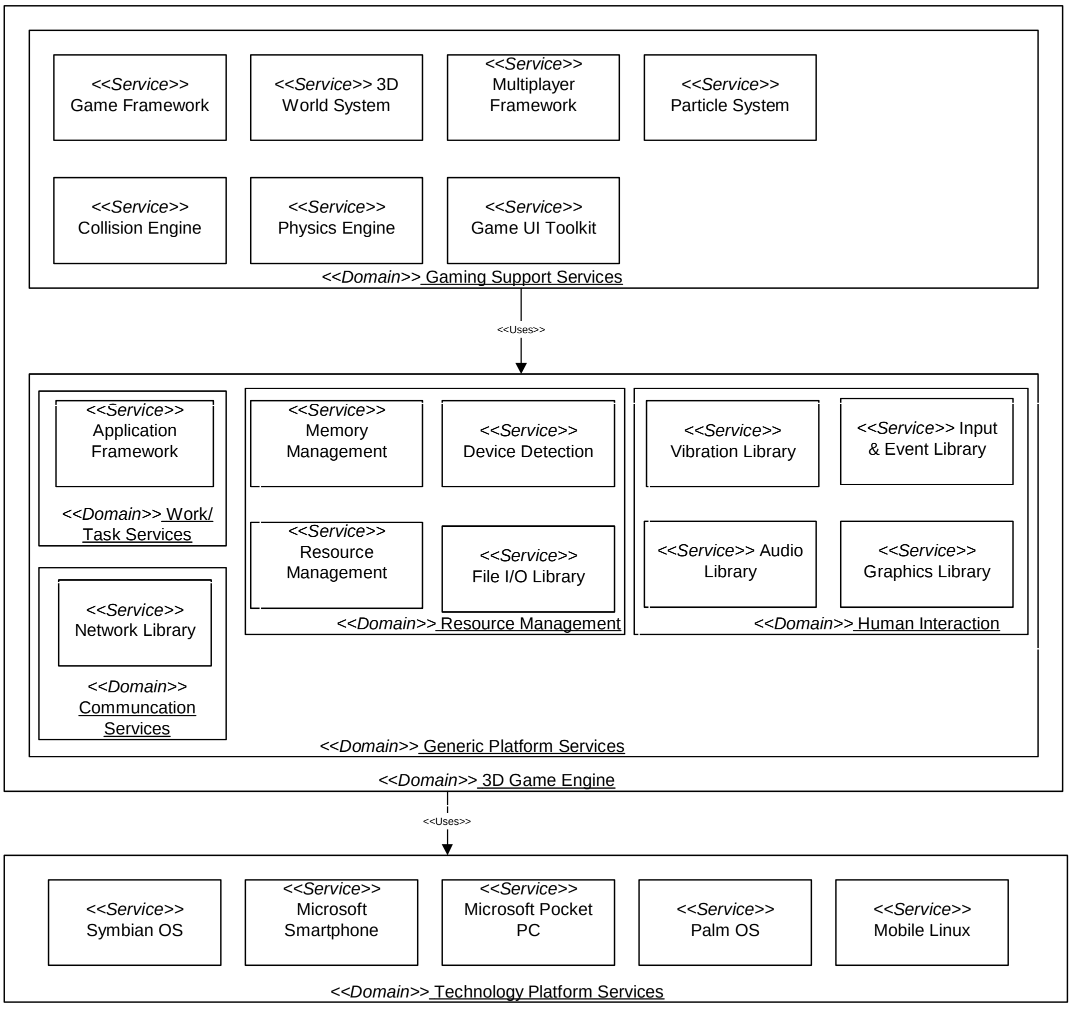

<<Service>> Game Framework
<<Service>> 3D World System
<<Service>> Multiplayer Framework
<<Service>> Particle System
<<Service>> Collision Engine
<<Service>> Physics Engine
<<Service>> Game UI Toolkit
<<Domain>> Gaming Support Services
<<Uses>>
<<Service>> Application Framework
<<Domain>> Work/ Task Services
<<Service>> Memory Management
<<Service>> Device Detection
<<Service>> Resource Management
<<Service>> File I/O Library
<<Domain>> Resource Management
<<Service>> Vibration Library
<<Service>> Input & Event Library
<<Service>> Audio Library
<<Service>> Graphics Library
<<Domain>> Human Interaction
<<Service>> Network Library
<<Domain>> Communcation Services
<<Domain>> Generic Platform Services
<<Domain>> 3D Game Engine
<<Uses>>
<<Service>> Symbian OS
<<Service>> Microsoft Smartphone
<<Service>> Microsoft Pocket PC
<<Service>> Palm OS
<<Service>> Mobile Linux
<<Domain>> Technology Platform Services

Figure 26. 3G game engine and related services of other domains.

Conceptual deployment of a service:

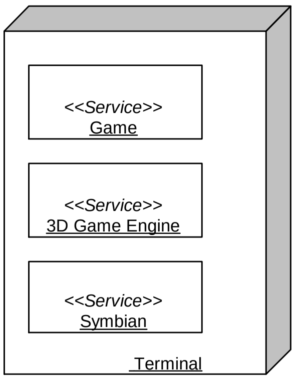


Figure 27. An example deployment of the 3G game engine.

External component diagram of the service:

Provided interfaces:

5.1.2 Game GUI library

Identification: A GUI library that extends J2ME functionality for gaming support domain.

Source:
Motorola, the concept with basic realization has been designed for WISE Pilot 2.
Java class path: com.motorola.microedition.ui.*

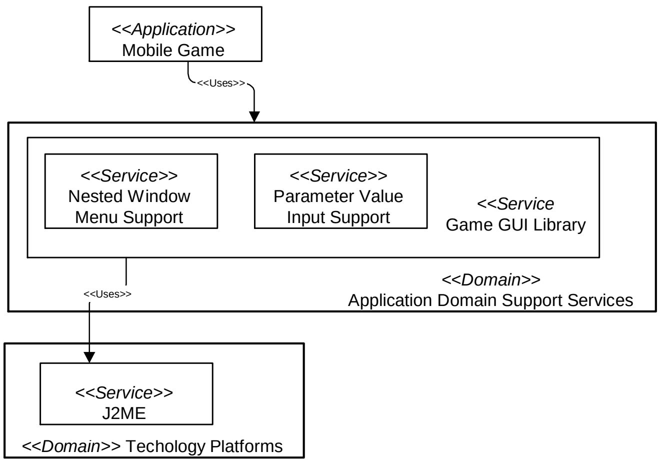
Overview:
This Java library provides a basic set of components to be used in J2ME-enabled heterogeneous clients in order to build generic menu-driven parameters settings.

Special terms and rules:
Due to its graphical approach, it's not suitable for device with very small screens (one or two line of text) where a text-based interface should be provided.

Features:
N/A

Conceptual structure

The library provides a basic framework for End-user input and display of values in a menu-driven context. For values we intend numbers, strings or a generic choice among, for example, pictures. Classes can easily be extended for application-specific attributes.



<<*Application*>> Mobile Game
<<Uses>>
<<*Service*>> Nested Window Menu Support
<<*Service*>> Parameter Value Input Support
<<*Service* Game GUI Library
<<*Domain*>> Application Domain Support Services
<<Uses>>
<<*Service*>> J2ME
<<*Domain*>> Techology Platforms

Figure 28: Conceptual structure of Game GUI Library

Conceptual Deployment

Game GUI is deployed in mobile terminal side.

External component diagram of the service:

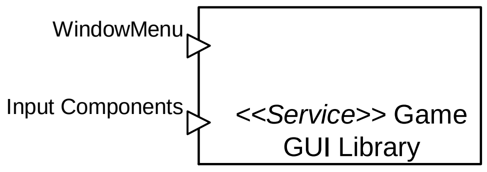


Figure 29. External interfaces of Game GUI Library.

Provided Interfaces

Windowmenu is a Java class that extends Java Canvas class with methods:

setAccept and *setCancel*, which provide the way out of the menu. Both require as parameters a displayable class and the display of the application.
goAccept and *goCancel* methods force the menu to close.
add method is called anytime a new component has to be added to the menu
The component added to menu must provide an *Appendable* Java interface

paint
keyPressed
pointerPressed
isMenu

Other methods are related to user interaction and are common Canvas methods of Java.

InputComponents is a set of Java components to handle

Adaptable input of parameter values in a range
Select picture among picture group
Get password etc. in an edit box.

5.1.3 Game Engine

TBD in iteration 3.

5.1.4 Location based services (Polos platform)

TBD in iteration 3.

5.1.5 Geographic services (GIS)

TBD in iteration 3.

5.2 GENERIC PLATFORM SERVICES

5.2.1 Negotiation protocol

TBD in iteration 3 (from Wise task 2.3 Agents).

5.2.2 Heterogeneous User Interface Service

Identification: Heterogeneous User Interface Service (HUS) belongs to the generic platform services.

Source: VTT, the concept with realization (made for the remote control of home appliances in ITEA/VHE project). Here HUS is adapted to the Wise context by replacing the OSGi server with the Wise Transport Service.

Overview:

HUS provides a generic mechanism to transform the presentation of a wireless service. The capabilities of wireless services and used terminals are described in the XML based UIML (User Interface Markup Language). HUS adapts (using the generic configuration service) the user interface of an application service to the format required by the user's preferences, the terminal capabilities and the features provided by the application.

Special terms and rules: HUS applies the broker pattern and has been implemented by Java.

Quality attributes:

The quality requirements set to HUS and how these requirements are intended to be met are defined in Table 9.

Table 9. Quality requirements and their intended realization.

Quality requirement	**Description**	**Realization**
Portability	• HUS can be used with different kinds of technology platforms (i.e. lower level protocols, implementation languages, operating systems etc.)	• HUS uses the technology platform services through proxies that need to be redeveloped if the technology platform is changed.
Modifiability	• The use of new types of applications and terminals requires a minimal work if any.	• The transcoders and the feature descriptions of applications can be updated.
Extendibility	• New terminal types can be added to the wireless network. • A new type of application service could register and use the terminals available in the network.	• Easily linked by using UIML descriptions, transcoders and the configuration service.
Reusability	• HUS can be applied to any kind of application through a particular service interface.	• HUS provides a generic interface to the wireless service to be connected to the system. • HUS requires that all application services are designed according to the used pattern.

Features:
UIML_Service (the core of the broker):
- Communicates with
  1) applications through the Wise Transport Service,
  2) transcoders internally through specified interfaces, and

3) client devices through an external HTTP Server.
- Co-ordinates the necessary transcoding process by exploiting the capabilities of UIML_Transcoders.

UIML transcoders
- provide resources for UIML_Service, including one or more transcoders (that transform data from UIML to the destination languages),
- can be updated and new ones can be added, and
- use the configuration service.

Applications must
- register themselves as services,
- implement the UIML_Application_Interface, and
- provide identification information of each application.

- Configurable features:
  - user preferences
  - user interface (browser) capabilities

Capabilities of browsers:
- most typical HTML and VXML browsers used by client proxies to recognize client formats.

Current implementation uses an HTTP server in communication between browsers and UIML_Service.

Wise Transmit Service provides message and event based communication between application services and HUS.

Conceptual structure:

Figure 30 presents the conceptual structure of the HUS service.

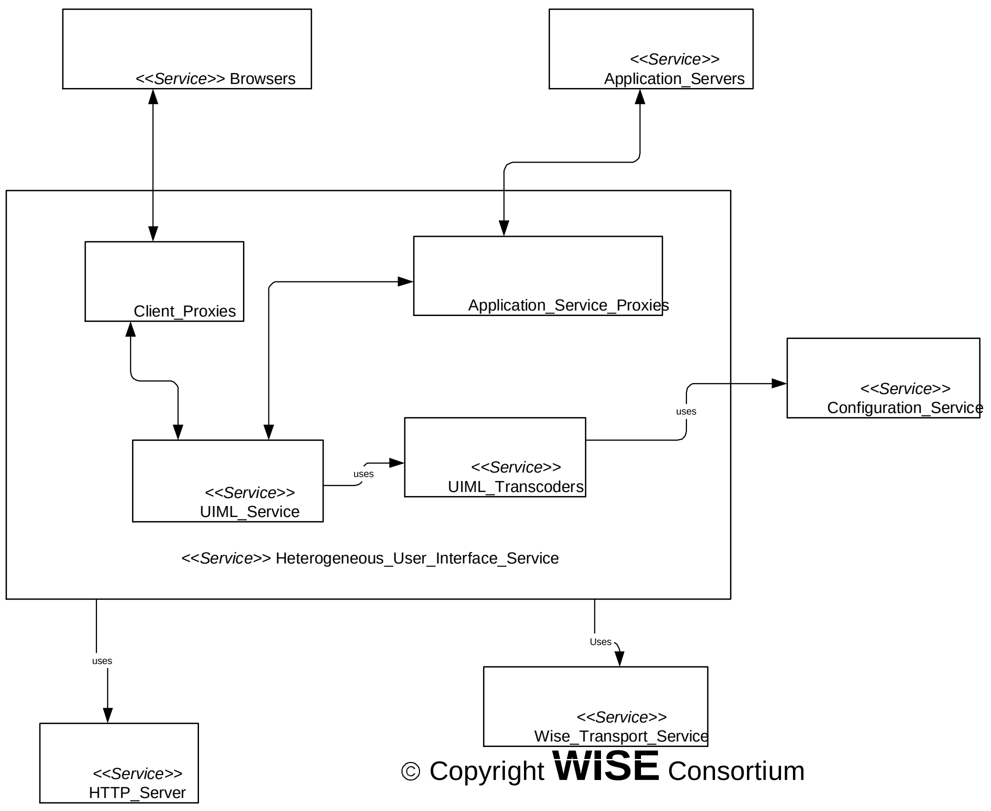

<<*Service*>> Browsers
<<*Service*>> Application_Servers
Client_Proxies
Application_Service_Proxies
<<*Service*>> Configuration_Service
uses
<<*Service*>> UIML_Transcoders
uses
<<*Service*>> UIML_Service
<<*Service*>> Heterogeneous_User_Interface_Service
uses
Uses
<<*Service*>> Wise_Transport_Service
<<*Service*>> HTTP_Server

Figure 30. Conceptual structure of HUS.

Table 10. Responsibilities of the HUS components.

Component	**Responsibilities**
HUS	Provides a uniform communication channel between any kind of user interface and applications. Self-configures operation according to the used terminal and the capabilities of the application service.
UIML_Service	The broker that mediates messages and controls the transformation process.
UIML_Transcoder	Transforms the format of an application to the form of the terminal browser.
Client_Proxy	Separates messaging from connections between the broker and clients. A connection point for a client.
Application_Service_Proxy	Represents an application service to the broker providing a link to the definition of the application UI.

Conceptual deployment:

The HUS service can be deployed in several ways. Figure 31 shows one deployment of the HUS service to a wireless service environment separating applications and generic platform services to different nodes.

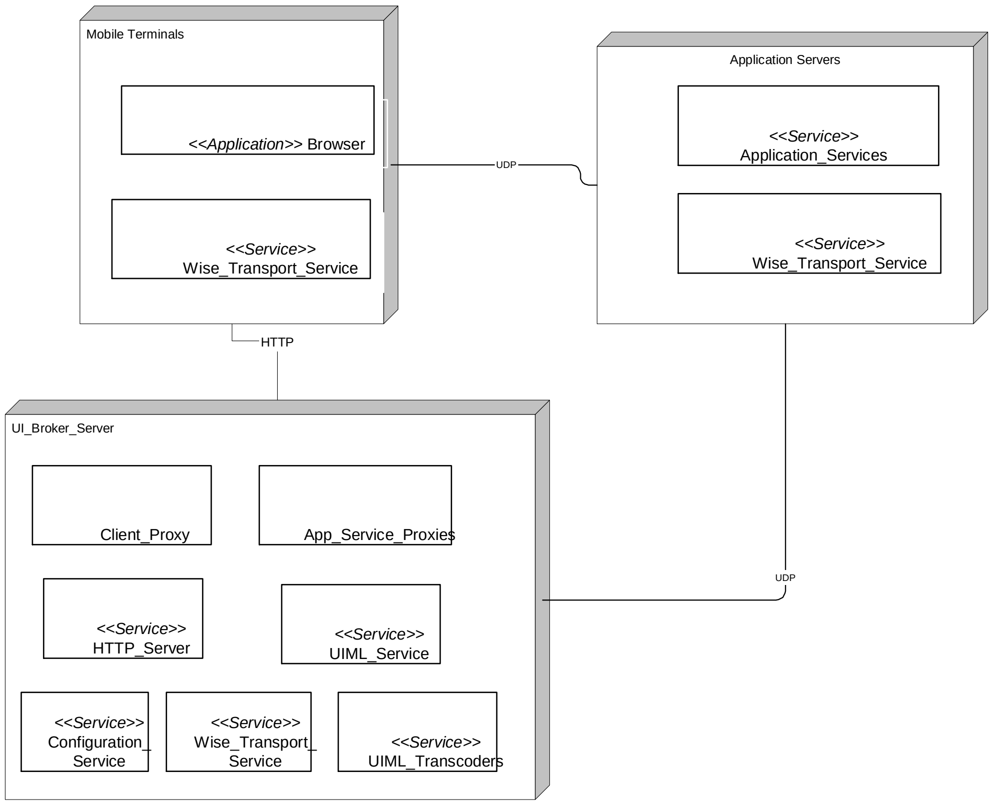

Mobile Terminals
<<Application>> Browser
<<Service>> Wise_Transport_Service
UDP
Application Servers
<<Service>> Application_Services
<<Service>> Wise_Transport_Service
HTTP
UI_Broker_Server
Client_Proxy
App_Service_Proxies
<<Service>> HTTP_Server
<<Service>> UIML_Service
UDP
<<Service>> Configuration_Service
<<Service>> Wise_Transport_Service
<<Service>> UIML_Transcoders

Figure 31. An example deployment of the HUS service.

External component diagram:

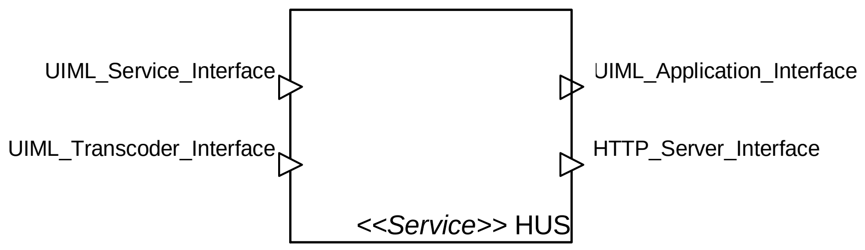


Figure 32. External interfaces of HUS.

Provided interfaces:

UIML_Service_Interface provided by UIML_Service.

- Push_UI_Description
- Acquire_Value

UIML_Transcoder_Interface provided by each transcoder.

Required interfaces:

UIML_Application_Interface provided by each application:

- Retrieve_UI_Description
- Request_Value
- Invoke_Method
- Enable/Disable_Push_Service

HTTP_Server_Interface (HTTP_GET):

- UI_Homepage
- Application_List
- UI_Action

Wise Transport Service is used to connect applications to the UIML_Service and Configuration Service.

5.2.3 Wise Transport service

Identification:
Wise Transport Service (WTS) provides generic message-based communication service, supporting both synchronous and asynchronous modes. It is based on UDP Service.

Source:
Motorola and Sodalia, developed in Wise.

Special terms and rules:
Wise Transport Service utilizes the client-server architecture style.

Quality attributes:

Requirement	Definition	Realization
Efficiency	Wise Transport service should provide message transport in close to real-time.	UDP provides better efficiency than TCP. (Estimated less than ¼ Round trip time)
Scalability	The client side of the service should fit (in size) into the mobile terminal.	The client side functionality is kept as simple as possible.

Features:

Communication

Synchronous

Asynchronous

Reliable

Unreliable

// Synchronous/asynchronous concepts are strictly related to reliable/unreliable packets.

Communication Protocol

UDP

Operating Environment

J2ME

J2EE

Conceptual structure:

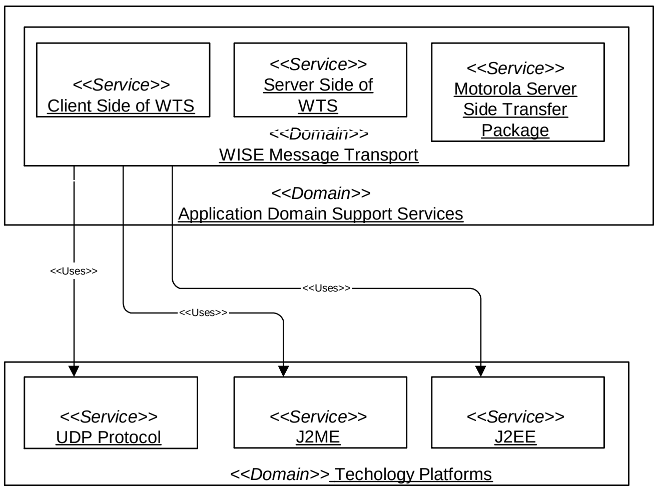

<<Service>> Client Side of WTS
<<Service>> Server Side of WTS
<<Service>> Motorola Server Side Transfer Package
<<Domain>> WISE Message Transport
<<Domain>> Application Domain Support Services
<<Uses>>
<<Uses>>
<<Uses>>
<<Service>> UDP Protocol
<<Service>> J2ME
<<Service>> J2EE
<<Domain>> Techology Platforms

Figure 33. The conceptual structure of WTS.

Conceptual deployment:

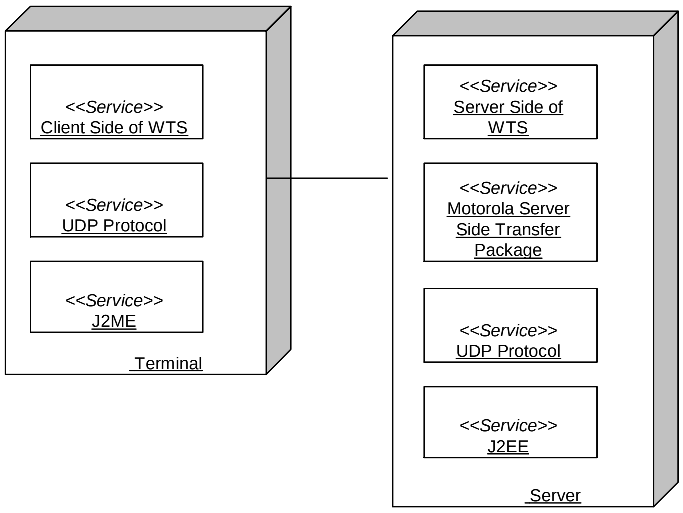

<<Service>> Client Side of WTS
<<Service>> UDP Protocol
<<Service>> J2ME
Terminal
<<Service>> Server Side of WTS
<<Service>> Motorola Server Side Transfer Package
<<Service>> UDP Protocol
<<Service>> J2EE
Server

Figure 34. An example deployment of WTS.

External component diagram of the service:

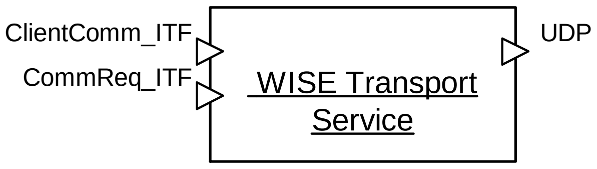


Figure 35. External interfaces of WTS.

Provided interfaces:
ClientComm_ITF
- Send
- SendReliable
- Receive
- hasMoreData
- Stop

CommReq_ITF
- ..

Required interfaces:
UDP
- .

5.2.4 Multimedia streaming service (MMS)

Identification:
The multimedia service is capable of streaming multimedia (e.g. video and/or sound) bi-directionally between two terminals.

Source:
The service has been introduced in [54]. The service is commercially available from Hantro (http://www.hantro.com).

Special terms and rules:
Multimedia streaming service utilizes blackboard architecture style and layered style.

Quality attributes:

Requirement	Definition	Realization
Modifiability	Easy and flexible adding (e.g. QoS protocols) and modification of the features should be considered.	• Selected architecture style (blackboard) allows easy modifications and additions.
Integrability	The platform is intended to be part of a wider multimedia platform rather than a stand-alone application. Thus major emphasis should be placed on the ease of integrating the platform into the existing products.	• This requirement is notified in the interface design.
Portability	The platform should be easy to port to different operating environments.	• Platform depended code is isolated with compiler flags.

Features:
Multimedia streaming
- Sending
- Receiving

Signaling Protocol
- SIP
  - Calls via proxy
  - Direct Calls

Transfer Protocol
- RTP

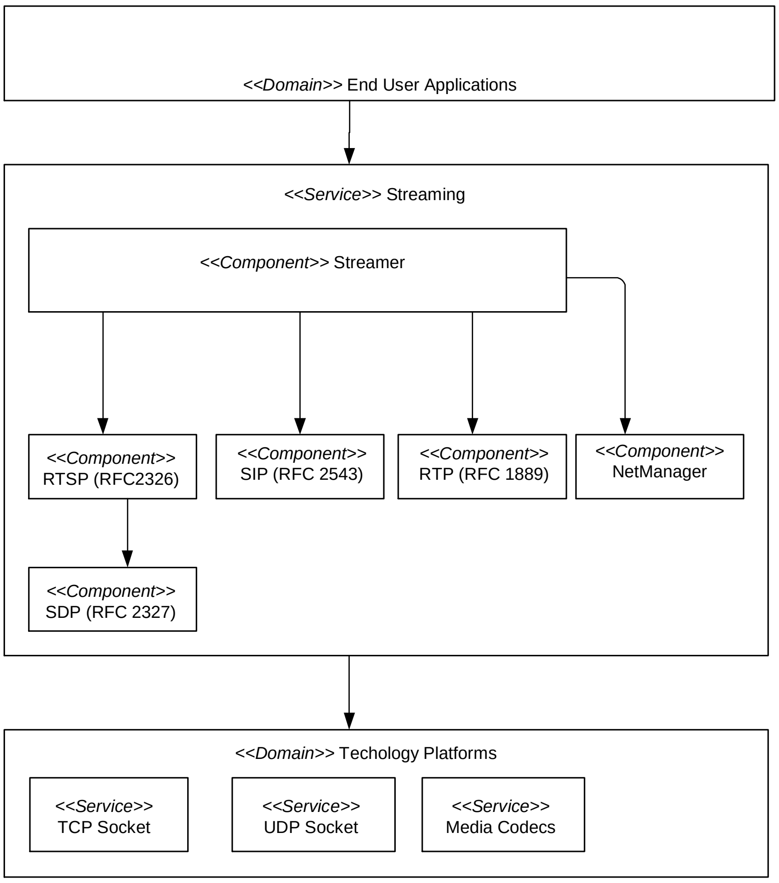
Stream Control Protocol
- RTSP

Programming Language
- C

Communication protocol
- TCP Sockets
- UDP Sockets

Operating System
- WinCE
- Windows
- Epoc

Conceptual structure:



<<*Domain*>> End User Applications
<<*Service*>> Streaming
<<*Component*>> Streamer
<<*Component*>> RTSP (RFC2326)
<<*Component*>> SIP (RFC 2543)
<<*Component*>> RTP (RFC 1889)
<<*Component*>> NetManager
<<*Component*>> SDP (RFC 2327)
<<*Domain*>> Techology Platforms
<<*Service*>> TCP Socket
<<*Service*>> UDP Socket
<<*Service*>> Media Codecs

Figure 36. Conceptual structure of MMS.

Table 11. Responsibilities of the elements of MMS.

Component	**Responsibilities**
Streamer	Streamer component controls the Streaming service. It handles interaction with the upper and lower level services or corresponds the data repository in the blackboard architecture style.
RTSP	Implements RTSP-protocol (Real-time Streaming Protocol ; IETF RFC 2326). RTSP provides session control for RTP streams.
SIP	Implements SIP-protocol (Session Initiation Protocol ; IETF RFC 2543). SIP is a signaling protocol for creating, modifying and termination sessions i.e. calls between one or more participants.
RTP	Implements RTP-protocol (Real-time Protocol ; IETF RFC 1889). RTP provides end-to-end network transport function suitable for applications transmitting real-time data.
NetManager	Provides an interface to the network.
SDP	Dummy implementation of SDP protocol (Session Description Protocol ; IETF RFC 2327). SDP is used to describe media types and parameters used in RTP

	streams.

Conceptual deployment:

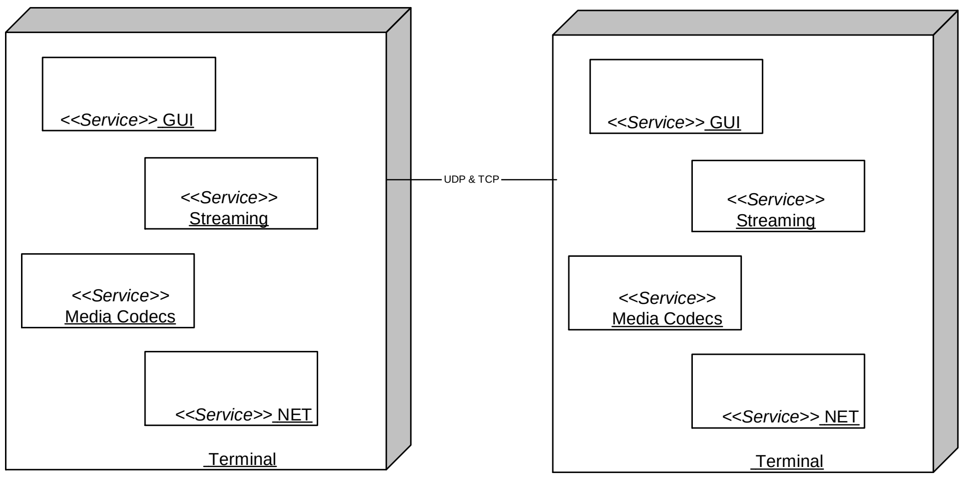


Figure 37. An example deployment of MMS.

External component diagram:

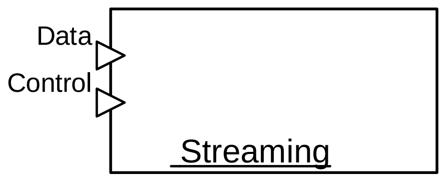


Figure 38. External interfaces of MMS.

Provided interfaces:
Control

Initialize
Start stream
Stop stream

Required interfaces*:*
Net

UDP Socket Interface
TCP Socket Interface

Codec

Encode data
Decode data

5.2.5 Instant Messaging & Presence Service

Identification:

Purpose of this service is to provide instant messaging and presence service (IM/P) for the applications. Instant messaging is transferring messages between users in near real-time. The messages are usually, but not required to be, short media messages, preferably text. The Presence service is defined as a subscription to and notification of changes in the communication state of a user.

Source:
The service has been introduced in [54]. This service is commercially available from Creanor (http://www.creanor.com).

Special terms and rules:
IM/P service utilizes blackboard architecture style and layered style.

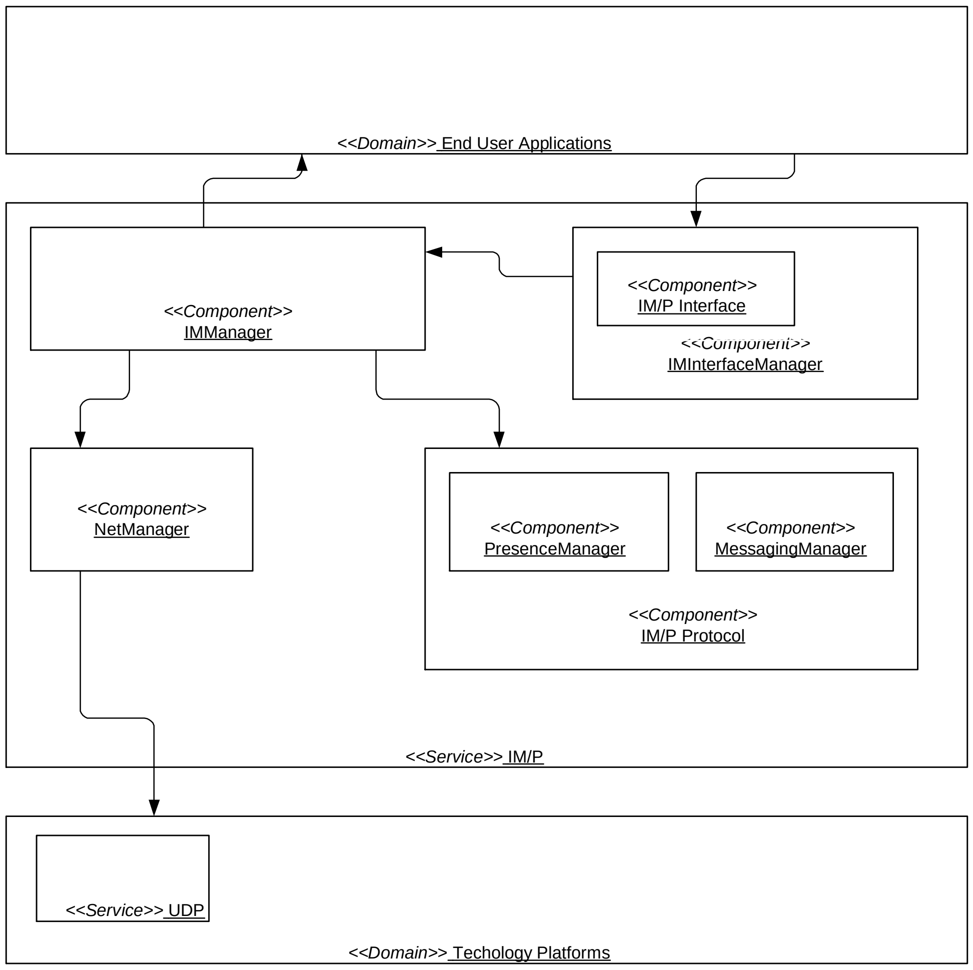
Quality attributes:

Table 12.Quality requirements and their intended realizations.

Requirement	Definition	Realization
Modifiability	• Architecture should enable the flexible integration of new instant messaging protocols. • The interface between the service platform and GUI is not fixed to any implementation technique. • Architecture should support both the two presence server location types: local and remote Presence Agents (PAs).	• Addition of new protocols has been notified in the design of provided interfaces and in the implementation of the generic components.
Integrability	• The platform is intended to be usable in stand-alone application as well as a part of a wider multimedia platform.	• The requirement has been notified in the interface design.
Portability	• Platform should be easy to port to different operating environments, graphical user interfaces or network implementations.	• Platform depended code is isolated with compiler flags.

Features:
- Presence Protocol
  - SIMPLE
    - Local Presence Agent
    - Remote Presence Agent
  - Wireless Village (currently not implemented)
- Messaging Protocol
  - SIMPLE
    - Messages via Proxy
    - Direct messages
  - Wireless Village (currently not implemented)
- Programming Language
  - C
- Communication protocol
  - UDP Sockets
- Operating System
  - Windows
  - Linux

Conceptual structure:



<<Domain>> End User Applications
<<Component>> IMManager
<<Component>> IM/P Interface
<<Component>> IMInterfaceManager
<<Component>> NetManager
<<Component>> PresenceManager
<<Component>> MessagingManager
<<Component>> IM/P Protocol
<<Service>> IM/P
<<Service>> UDP
<<Domain>> Techology Platforms

Figure 39. Conceptual structure of IM/P.

Table 13. Responsibilities of IM/P components.

Component[1]	**Responsibilities**
IM/P Interface	Variable component that provides interface between IM domain and the GUI. Interface may be realized with C library or OS-Messaging.
IMInterfaceManager	Provides to the IM domain generic interface, which is used by specified IM/P interface component.
IMManager	Controller part of the IM domain. Is responsible to construct the system, control network connections, receive requests from the user and control the protocol components.
NetManager	Provides an interface to the network.
IM/P Protocol	Variable component which provides instant messaging and presence functionality. Protocol may either be SIP or WV.
PresenceManager	Variable component inside IM/P component, that provides control for presence. Functionality depends on the selected protocol (SIP/WV).

[1] Variable components are marked with gray background.

	Functionality depends on the selected protocol (SIP/WV).
MessagingManager	Variable component inside IM/P component, that provides control for messaging. Functionality depends on the selected protocol (SIP/WV).

Conceptual deployment:

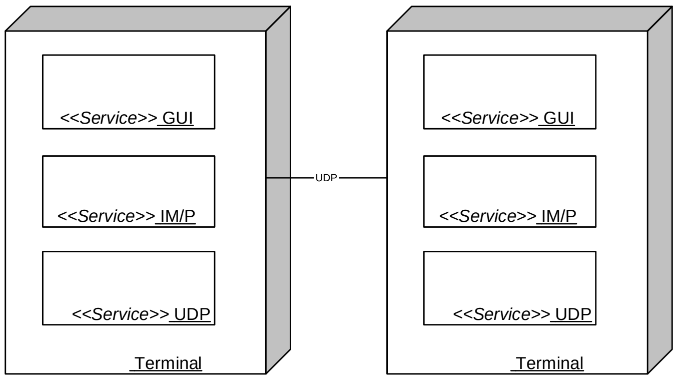


Figure 40. Conceptual deployment of IM/P.

External component diagram:

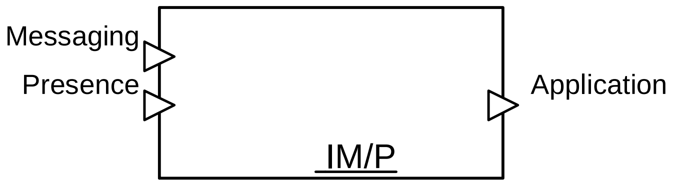


Figure 41. External interfaces of IM/P.

Provided interfaces:
Messaging:
  Send Message
Presence:
  Subscribe buddy
  Cancel subscription
  Notify

Required interfaces:
Net:
  UDP Socket Interface
Application:
  Handle incoming message
  Handle incoming notification

5.2.6 Configuration Service

Identification:

Configuration service (CS)is used to configure the other services according to the feature models about the service, the user choices and the environment features.

Source: VTT.

Special terms and rules:

Quality attributes:

Table 14. Quality requirements and their intended realization.

Requirement	Definition	Realization
Usability	The interface has to be a simple enough. Feature model and profile information has to be in human-readable format.	The interface two abstraction levels – high level for users and low level for developers.
Scalability	The client side of the service should fit (in size) into the mobile terminal.	The client side functionality is kept as simple as possible.

Features:

- Feature Model
  - Local
  - In the Network
  - Format
    - Custom
    - RDF
- Profile Information
  - Local
  - In the Network
  - Format
    - Custom
    - RDF

Conceptual structure:

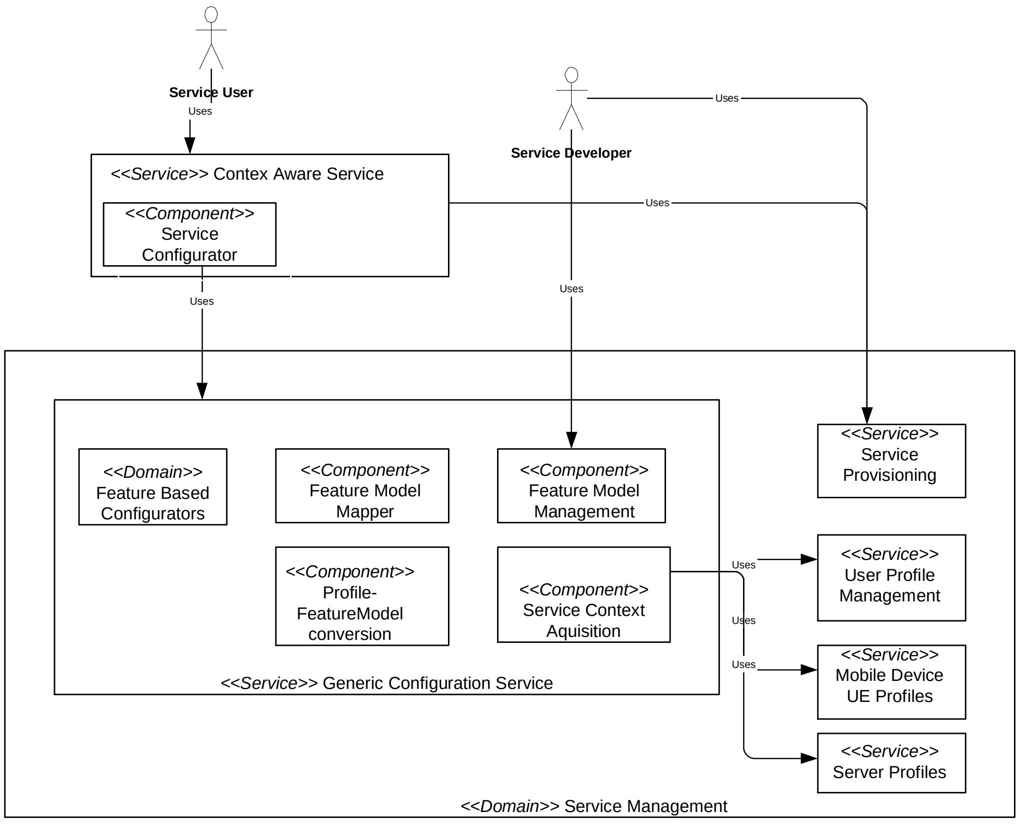

Service User
Uses
Service Developer
Uses
<<Service>> Contex Aware Service
<<Component>> Service Configurator
Uses
Uses
Uses
<<Domain>> Feature Based Configurators
<<Component>> Feature Model Mapper
<<Component>> Feature Model Management
<<Component>> Profile-FeatureModel conversion
<<Component>> Service Context Aquisition
<<Service>> Generic Configuration Service
<<Service>> Service Provisioning
Uses
<<Service>> User Profile Management
Uses
Uses
<<Service>> Mobile Device UE Profiles
<<Service>> Server Profiles
<<Domain>> Service Management

Figure 42. Conceptual structure of CS.

Table 15. Responsibilities of the CS components.

Component	Responsibilities
Context Aware Service	Any service that should adapt to context requirements
Service Configurator	Component of a service that is responsible for configuration/adaptation of it
Service Management	Domain responsible of various management functions related to wireless services
Generic Configuration Service	Provides an interface to generic feature based configuration service for context aware services (configuration of sub-services?)
Feature Model Management	Provides services needed to create feature models and provides storage facilities for them
Feature Based Configurator	A configurator service that takes a set of feature selections as input and produces specific configuration information as output. The configurator could use: 1. Mapping of a set of feature selections into set of service specific features based on a set of rules (uses feature selection service). 2. Provides a set of default features based on the feature model and adds them to feature selections 3. Modification of service metamodel based on a set of features to be used by reflection pattern 4. Generation of a part of service based on the set of features. 5. ...
Feature Selection	Provides an interface for selecting features of a feature model. Checks that a set of selections is valid against the restrictions set by the feature model.
Service Context Acquisition	Provides an interface for accessing context information from various sources in a standard way
Profile-Feature Conversion	Converts profile information from various sources into feature selections and vice-versa
Service Provisioning	Service downloading and subscription etc..
User Profile Management	Stores service user profiles and provides an interface for accessing them.
Mobile Device UE Profiles	Stores profile information about various mobile devices.
Server Profiles	Stores profile information about servers that run wireless services and provides interface for accessing the information

Conceptual deployment:

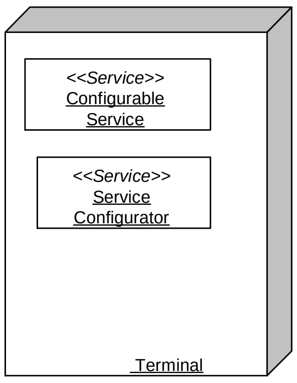


Figure 43. An example deployment of CS.

External component diagram:

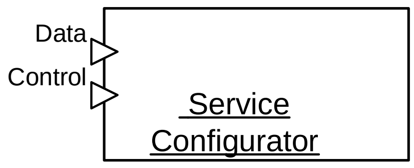


Figure 44. External interfaces of CS.

Provided interfaces:
Data:
    Initialize feature models
    Initialize profiles
Control:
    Configure

Required interfaces:

5.2.7 Data Management Component

Identification: Provides wireless data management services in WISA generic support services domain.

Source:

Contains COTS database component by Solid.

Overview:

Data management component provides basic data storage, transfer, synchronization and management services for wireless services and its management services. A wireless service adapts the data management component for its purposes using standard database interfaces and defining publications and procedures for data synchronization.

Special terms and rules:

TBD in iteration 2.

Quality attributes:

The quality requirements set to DMC and how these requirements are intended to be met, are defined in Table 16 (TBD in iteration 2).

Table 16. Quality requirements and their intended realization.

Quality requirement	Description	Realization
Portability		
Modifiability		
Extendibility		
Reusability		

Features:
Operating Systems
Conceptual structure:

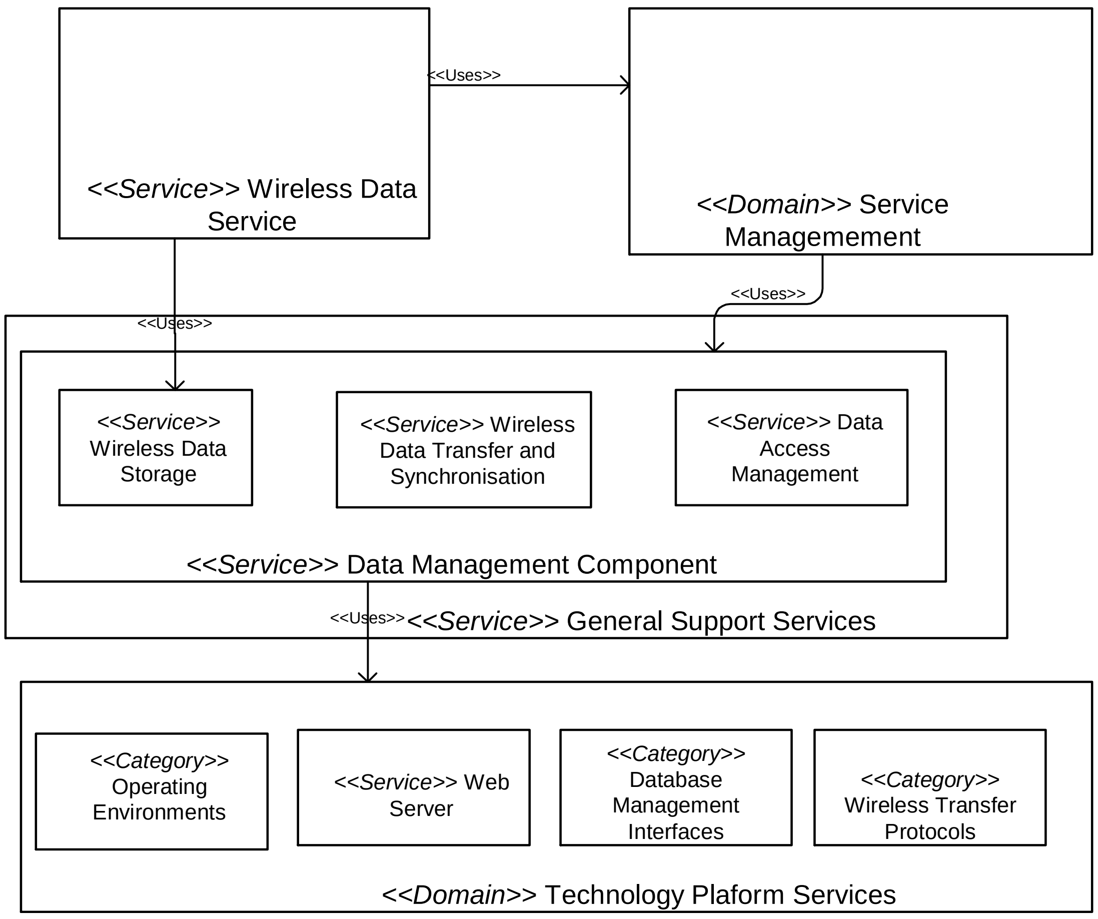

<<Service>> Wireless Data Service
<<Uses>>
<<Domain>> Service Managemement
<<Uses>>
<<Uses>>
<<Service>> Wireless Data Storage
<<Service>> Wireless Data Transfer and Synchronisation
<<Service>> Data Access Management
<<Service>> Data Management Component
<<Uses>>
<<Service>> General Support Services
<<Category>> Operating Environments
<<Service>> Web Server
<<Category>> Database Management Interfaces
<<Category>> Wireless Transfer Protocols
<<Domain>> Technology Plaform Services

Figure 45. Conceptual Structure of DMC.

Table 17. Conceptual Elements of Data Management Component.

Conceptual Element	Description
Wireless Data Storage	Provides a data definition, storage and access service for both local and networked data.
Data Transfer and Synchronization	Provides service to define transferring and replication data between local and networked data storage for wireless systems.
Data Access Management	Provides database access management services

Conceptual deployment:
N/A

External component diagram:
N/A

Provided interfaces:

API: (SQL, ODBC and JDBC)

Service defines database tables and Procedures and Publications for data transfer and synchronization using the API

5.3 TECHNOLOGY PLATFORM SERVICES

5.3.1 WAP

Identification: A specification in Software Environments domain for wireless services.

Source:

http:\\www.openmobilealliance.org

Overview:

The Wireless Application Protocol (WAP) is a result of continuous work to define an industry wide specification for developing applications that operate over wireless communication networks. The WAP programming model is the WWW programming model with a few enhancements.

5.3.2 J2SE

Identification: A standard in Software Environments domain for wireless services.

Source:

http://java.sun.com/j2se/

Overview:

Java ™ 2 Platform, Standard Edition is the standard platform for Java ™ 2.

5.3.3 J2EE

Identification: A standard in Software Environments domain for wireless services, server side.

Source:

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

Overview:

The Java™ 2 Platform, Enterprise Edition (J2EE) defines the standard for developing multitier enterprise applications. Compared to J2SE, Enterprise Edition adds full support for Enterprise JavaBeans™ (EJB) components, Java Servlets API, JavaServer Pages™ (JSP) and XML technology.

5.3.4 J2ME

Identification: A standard in Software Environments domain for wireless service, mobile terminals.

Source:

http://java.sun.com/j2me/

Overview:

Java ™ 2 Platform, Micro Edition, J2ME is an optimized Java runtime environment. Currently there are two J2ME configurations: the Connected Limited Device Configuration (CLDC) and the Connected Device Configuration (CDC). A configuration is comprised of a virtual machine, core libraries and APIs. CDLC is designed for devices with constrained CPU and memory resources. CDC is designed for next-generation devices with more robust resources.

Features:
Configuration
- CLDC
- CDC

5.4 SERVICE MANAGEMENT SERVICES

5.4.1 Service Management Component (SMC)

Identification: Provides several services in WISA Service Management domain.

Source:

Contains COTS components by Sodalia and components, facades and clients developed in WISE.

Overview:

The Service Management Component (SMC) is a provider of service management services to accomplish a common set of operations required in providing a service. The future development of service management component in Wise will concentrate on:
- Additional wireless specific services such as Location Services, etc.
- The Service Management Proxy, which is the value-adding component of WISA for service management. Due to its easy-to-use interface, it allows for rapid development of a service application hiding all the details related to the Service Management domain.
- Stardard interfaces [e.g. OSS through Java] used by the Service Management Proxy to access the real services
- Service Management Products that could be integrated with the Service Management Proxy

Special terms and rules:

SMC is a separate service management server with façade interface provided to the application servers. WAP and HTML interfaces are provided for end-user and administrator operations.

Client-Server and Proxy and Facade patterns are used to integrate the COTS components that reside in the service management server to WISA.

Quality attributes:

The quality requirements set to SMC and how these requirements are intended to be met, are defined in Table 18 (TBD in iteration 2).

Table 18. Quality requirements and their intended realization (TBD)

Quality requirement	Description	Realization
Portability		
Modifiability		
Extendibility		
Reusability		

Features:

Technology for service subscription interface:
- WAP
- HTML

Conceptual structure:

Figure 46 presents the conceptual structure of the SMC component. The descriptions of actors are in Table 1 and the responsibilities of conceptual elements in Table 2.

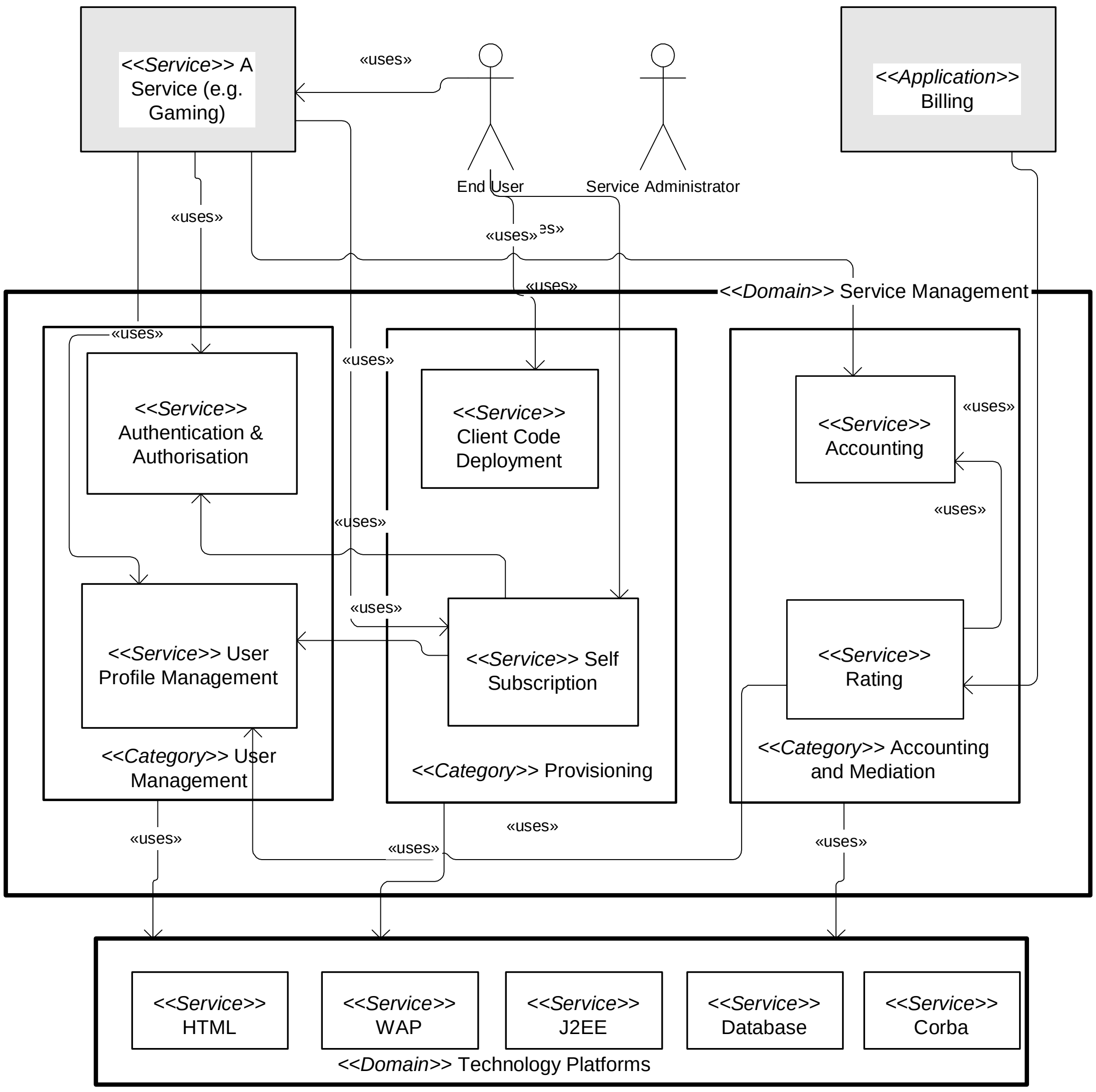

<<*Service*>> A Service (e.g. Gaming)
«uses»
End User
Service Administrator
<<*Application*>> Billing
«uses»
«uses»
«uses»
«uses»
<<*Domain*>> Service Management
«uses»
«uses»
<<*Service*>> Authentication & Authorisation
<<*Service*>> Client Code Deployment
<<*Service*>> Accounting
«uses»
«uses»
«uses»
«uses»
<<*Service*>> User Profile Management
<<*Service*>> Self Subscription
<<*Service*>> Rating
<<*Category*>> User Management
<<*Category*>> Provisioning
<<*Category*>> Accounting and Mediation
«uses»
«uses»
«uses»
«uses»
<<*Service*>> HTML
<<*Service*>> WAP
<<*Service*>> J2EE
<<*Service*>> Database
<<*Service*>> Corba
<<*Domain*>> Technology Platforms

Figure 46. Conceptual structure of SMC.

Table 19. Actors of Service Management Component.

Conceptual Element	Description
End User	Uses a Service Uses the Client Code Download Service to download the client side code implementing the service (if the service requires a thick client) Uses the Self-Subscription Service to subscribe the service
System Administrator	Performs all the tasks needed to configure, monitor and update the service.

Table 20. Responsibilities of conceptual elements.

Conceptual Element	Responsibility
A Service Application (external)	Provides a service to the end user (e.g. trading online of stocks, gaming, etc.). Uses Authentication Service to give access only to granted users Uses User Profile Service to store user related data (what type of data is

	service dependant issue) Uses Usage Data Collection for charging the user for having used the service
Authentication Service	Provides the operations to grant the access only to authorised uses
User Profile Service	Provides the operations to define, store and retrieve user related data
Self Subscription Service	Provides the service provider the infrastructure to offer services online and users to subscribe online services by themselves. Uses Authentication Service to grant the subscribed user to use the service Uses User Profile Service to store user subscription data (e.g. credit card data, etc.) Uses the Service Application to notify that a user has subscribed the service
Client Code Download Service (provided by an external COTS)	Provides a user the ability to choose and download client code to access to a service
Accounting Service	Provides service applications the ability to log usage data to charge the user
Rating Service (provided by an external COTS)	Provides a rule based engine to apply billing strategies to usage data uses Authentication Service and User Profile Service to gather user information for rating purposes
Billing Service (provided by an external COTS, but out of scope of WISE project)	Provides the service operator with all the features to invoice the user for having used a service uses the Rating Service to receive rated accounting data about the usage of a service

Conceptual deployment:

The Mobile Device runs the WAP Browser, the client side code of the offered service and the Code Deployment Client. The Service Server hosts the server side part of the offered service and the Accounting Agent, which is in turn, the client side of the Accounting Service. The Management Node runs the Authentication and Authorization Server, the User Profile Server, the Client Code Deployment Server, the Self-Subscription Server, the Accounting Server (which is the server side of the Accounting Service) and the Rating Server. Communication between the nodes is based on TCP/IP, further details on used protocols are provided in the concrete architecture section.

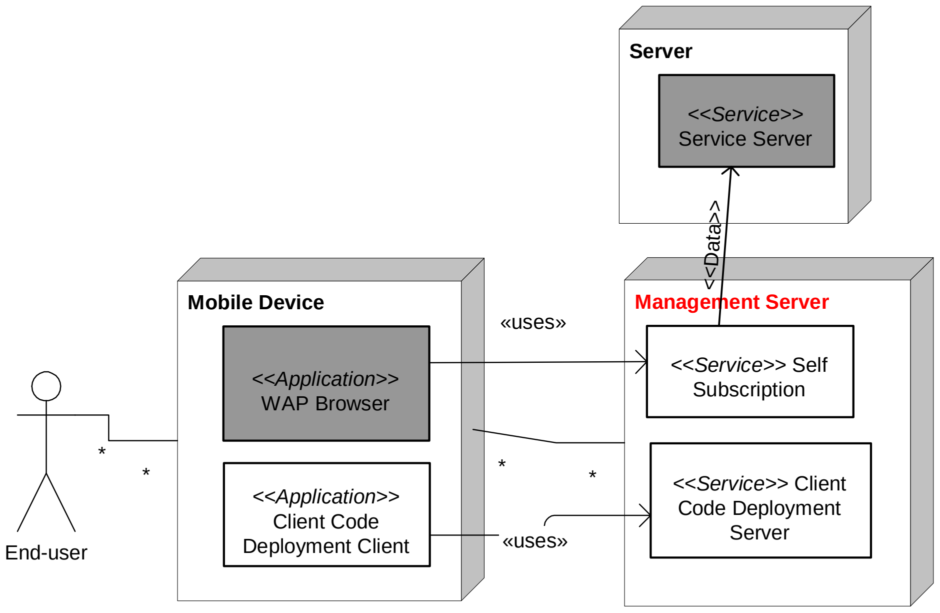

Server
<<Service>> Service Server
<<Data>>
Mobile Device
<<Application>> WAP Browser
<<Application>> Client Code Deployment Client
Management Server
<<Service>> Self Subscription
<<Service>> Client Code Deployment Server
«uses»
«uses»
*
*
*
*
End-user

Figure 47. Deployment of Provisioning Services.

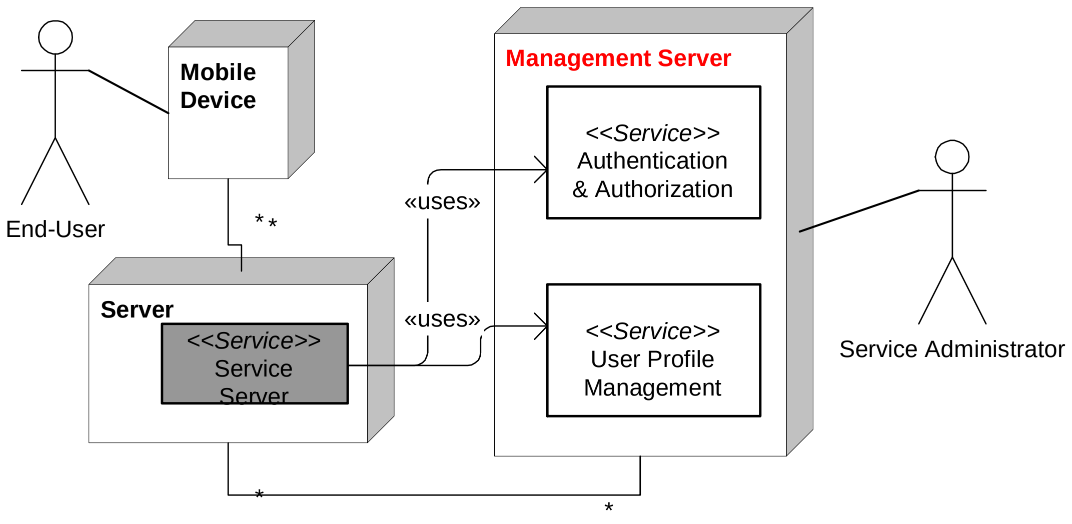

Mobile Device
Management Server
<<Service>> Authentication & Authorization
«uses»
End-User
* *
Server
<<Service>> Service Server
«uses»
<<Service>> User Profile Management
Service Administrator
*
*

Figure 48. Deployment of End-user Management Services.

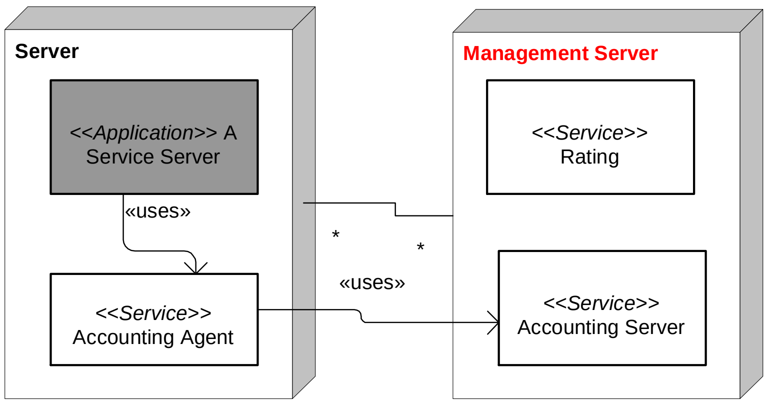

Server
<<Application>> A Service Server
«uses»
<<Service>> Accounting Agent
Management Server
<<Service>> Rating
*
*
«uses»
<<Service>> Accounting Server

Figure 49. Deployment Accounting and Meditation.

Both the Server Node and the Management Node could be split or replicated on more than one host depending on performance and fault tolerance requirements.

External component diagram:

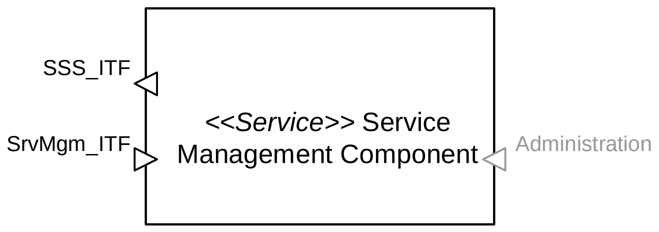


Figure 50. Interfaces of SMC provided to WISA.

Provided interfaces:

SrvMgm_ITF is a façade & Java Proxy for all service management interfaces

- Authentication
- User profile retrieval
- Accounting

Administration is several administrative interfaces to service management, which are not relevant for wireless service developer.

- End-user self-subscription interface
- Management of end-user authorization etc.

Required interfaces:

SSS_ITF is an JMS interface required from a server part of service that can be self-subscribed by end-user.

- Receive Activation

6. REFERENCES

[1] Agha, G. (2002). Adaptive Middleware, Communications of the ACM, 45, 6, pp. 31-32.

[2] Alur, D., Crupi, J., Malks, D. 2001. Core J2EE Patterns: Best Practices and Design Strategies. Prentice Hall. 496 p.

[3] Ammendola, G., Andreadis, A., Benelli, G., Giambene, G. 2002. Integration of distributed data sources for mobile services. Available on-line at: http://newton.ee.auth.gr/summit2002/ papers/SessionW2/2502128.pdf

[4] Bass, L., Clement, P. and Kazman, R. Software Architecture in Practice. Addison-Wesley. 1998.

[5] Borchers, J. 2001. A Pattern Apprach to Intgration Design, John Wiley.

[6] Buschmann, F. Building Software with Patterns, Proceedings of the Fourth European Conference on Pattern Languages of Programming and Computing, 1999, Bad Irsee, Germany, 58p.

[7] Buschmann, F., Meunier,R., Rohnert, H., Sommerlad, P., Stal, M. 1996. Pattern-oriented software architecture, a system of patterns. John Wiley & Sons. 457 p.

[8] Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R., Stafford, J., 2002. Documenting Software Architectures- Views and Beyond. Addison-Wesley. ISBN: 0-201-70372-6. 560 p.

[9] Corsaro, A., Schmidt, D., Klefstad, R. O'Ryan, C. 2002. Virtual Component, A Design Pattern for Memory-Constrained Embedded Applications. Proceedings of the 9th Conference on Pattern Language of Programs, PLoP 2002, Monticello, Illinois, September 8th-12th, 2002, 13 p.

[10] Cross, J., Schmidt, D. 2002. Quality Connector, A Pattern Language for Provisioning and Managing Quality-Constrained Services in Distributed Real-time and Embedded Systems. Submissions to the OOPSLA 2002 workshop 'Patterns in Distributed Real-time and Embedded Systems', Seattle, Washington, USA, November 5, 2002. 19 p.

[11] Cugola, G., Di Nitto, E., Fuggetta, A., “Exploiting an event-based infrastructure to develop complex distributed systems”, In 20th International Conference on Software Engineering, 1998.

[12] Dailey, H. PIECING IT TOGETHER, New J2EE Patterns Catalog Helps Solve the J2EE Architecture Puzzle. http://java.sun.com/features/2001/03/patterns.html.

[13] Dobrica, L., Niemelä, E. 2002. A Survey on Software Architecture Analysis Methods. IEEE Transactions on Software Engineering, Vol. 28, No 6, July 2002. pp. 638-653.

[14] Fisher, G. E. Guide on Opens System Environment (OSE) Procurements. NIST Special Publication 500-220. Oct. 1994.

[15] Florijn, G. Architectural styles and patterns. Available on-line at: http://www.cs.uu.nl/docs/vakken/swa/Slides/SA-5-styles.pdf. Referenced: (3.12.2002)

[16] Gamma, E., Helm, R., Johnson, R. and Vlissides, J. 1995). Design patterns: Elements of Reusable Object-Oriented Software, New York, Addison-Wesley, 395p.

[17] Garbinato, B., Guerraoui, R. 1997. Using the Strategy design pattern to compose reliable distributed protocols. Proceedings of the 3rd Conference on Object-Oriented Technologies and Systems (COOTS-3).

[18] Giese, H., 2001.Design Pattern and Software Architecture: Software Architecture. Available on-line at: http://www.uni-paderborn.de/cs/ag-schaefer/Lehre/Lehrveranstaltungen/Gill, C., Niehaus, D., DiPippo, L., Wolfe, V., Welch, L. 2002. Mapping a Multi-Level Scheduling Pattern Language to Distributed Real-Time Embedded Applications. Submissions to the OOPSLA 2002 workshop 'Patterns in Distributed Real-time and Embedded Systems', Seattle, Washington, USA, November 5, 2002. 15 p.

[20] Gitsels, M., Sauter, J. 2000. Profile-based Service Browsing - A Pattern for Intelligent Service Discovery in Large Networks. Submissions to the OOPSLA 2000 workshop 'The Jini Pattern Language', Minneapolis, Minnesota USA, October 15-19, 2000. 3 p.

[21] Gnutella/Napster Comparison, URL http://www.gnutellanews.com/information/comparison.shtml.

[22] Gokhale, A. 2000. Patterns in Bluetooth. Submissions to the OOPSLA 2000 workshop 'The Jini Pattern Language', Minneapolis, Minnesota USA, October 15-19, 2000. 2 p.

[23] Gutberlet, L. ,2000. Peer-to-Peer Computing- A Technology Fad or Fact? European Business School/ Schloss Reichartshausen am Rhein. Term Paper of Information Systems Management Seminar.

[24] Hartman, R. 2001. Building on patterns. ADT may 2001. 9 p.

[25] Homayounfar, H., 2002**.** An advanced P2P architecture using autonomous agents. University of Guelph (Ontario, Canada). Master's Thesis. January 2002. 182 p.

[26] Kalaoja, J., Niemelä, E., Tikkala, A., Kallio, P., Ihme, T., Torchiano, M. WISA Reference Architecture, Deliverable ID: D4 (Part B).

[27] Kon, F., Costa, F., Blair, G. and Campbell, R. 2002. The Case for Reflective Middleware, Communications of the ACM, 45, 6, pp. 33-38.

[28] Laverty, R. Initial specification of framework and models, The ITEA project ROBOCOP (Robust Open Component Based Software Architecture for Configurable Devices Project), Deliverable 1.3, May 2002, http://www.extra.research.philips.com/euprojects/robocop/

[29] Lee, S., Han, D., Lee, D. 2000. A pattern for Managing Distributed Workflows. Proceeding of the 7th Conference on Pattern Languages of Programs (PloP 2000), Allerton Park, Monticello, Illinois, USA, August 13 - 16, 2000, 15 p.

[30] Lerner, M., Vaneck, G., Vidovic, N., Vrsalovic, D., 2000. Middleware Networks- Concept, Design and Deployment of Internet Infrastructure. Boston Kluwer Academic Publishers. 375 p.

[31] Longshaw, A. 2001. Choosing Between COM+, EJB, and CCM, In: Heineman, G. and Councill, W. (eds.). Component-Based Software Engineering, New York, Addison-Wesley, pp. 621-640.

[32] Loyall, J., Rubel, P., Schantz, R., Atighetchi, M., Zinky, J. 2002. Emerging Patterns in Adaptive, Distributed Real-Time, Embedded Middleware. Proceedings of the 9th Conference on Pattern Languages of Programs, PLoP 2002, Monticello, Illinois, September 8th-12th, 2002, 11 p.

[33] Lutz, J. 2000. EAI Architecture Patterns. EAI Journal, March 2000. Pp. 64-73.

[34] Marinucci, T., Welch, L., Masters, M., Werme, P. 2002. A Pattern Language for Engineering Dynamic Real-Time Applications. Submissions to the OOPSLA 2002 workshop 'Patterns in Distributed Real-time and Embedded Systems', Seattle, Washington, USA, November 5, 2002

[35] McLean, S. 2001. Tie Into Remote Objects, .NET Remoting and C# make it a snap to use the Observer design pattern in your distributed applications. .Net Magazine - Online, Available on-line at: http://www.fawcette.com/dotnetmag/2001_12/online/online_eprods/smclean/default.asp

[36] Messerschmitt, D. and Szyperski, C. 2001. Industrial and Economic Properties of Software: Technology, Processes, and Value, Berkeley, California, University of California at Berkeley, Computer Science Division, 51p. UCB//CSD-01-1130.

[37] Neil, M. 2001. Turning to OO Platforms During Mobile Phone Design. Available on-line at: http://www.commsdesign.com/story/OEG20011120S0075

[38] Niemelä, E., Matinlassi, M., Lago, P. 2002. Architecture-centric approach to wireless engineering. To be published in the book of IEC (International Engineering Consortium) 'Annual review of Communications', Vol. 46, June 2003.

[39] Olson, D. 2001. A pocket-sized Broker. In: Rising, L. (ed.). Design Patterns in Communications Software. New York, USA, Cambridge University Press, pp. 237-247.

[40] P2P Working Group, "Taxonomy of Peer-to-Peer Architectures" http://www.peer-to-peerwg.org/tech/taxonomy/Docs/P2P-Taxonomy-v095.doc (14.12.2002)

[41] Parameswaran, M.; Susarla, A.; Whinston, A.B., 2001. P2P Networking: An Information-Sharing Alternative. IEEE Computer, July 2001.Pärssinen, J., Turunen, M. 2000. Patterns for Protocol System Architecture. Proceeding of the 7th Conference on Pattern Languages of Programs (PloP 2000), Allerton Park, Monticello, Illinois, USA, August 13 - 16, 2000. 26 p.

[43] Pree, W. and Pasetti, A. 2001. Embedded Software Market Transformation Through Reusable Frameworks, In: Henzinger, T. and Kirsch, C. (eds.). Embedded Software, First International Workshop, EMSOFT 2001, Tahoe City, CA, USA, October 8-10, 2001, Proceedings, Berlin, Germany, Springer-Verlag, pp. 274 - 286.

[44] Pryce, N. 2001. Abstract Session: an object structural pattern. In: Rising, L. (ed.). Design Patterns in Communications Software. New York, USA, Cambridge University Press, pp. 191-208.

[45] Raatikainen, K. Middleware in Mobile World, OT Land, LogOn Technology Transfer GmbH, June, 2003.

[46] Schmidt, D. 2002. Middleware for real-time and embedded systems, Communications of the ACM, 45, 6, pp. 43-48.

[47] Schmidt, D., Cleeland, C. 2001. Applying a pattern language to develop extensible ORB middleware. In: Rising, L. (ed.). Design Patterns in Communications Software. New York, USA, Cambridge University Press, pp. 393-438.

[48] Schmidt, D., O'Ryan, C., Othman, O., Kuhns, F., Parsons, J.. 2001. Applying patterns to develop a pluggable protocols framework for ORB middleware. In: Rising, L. (ed.). Design Patterns in Communications Software. New York, USA, Cambridge University Press, pp. 439494.

[49] Schmidt, D., Stal, M., Rohnert, H. and Buschmann, F. 2000. Pattern-Oriented Software Architecture, Volume 2: Patterns for Concurrent and Networked Objects, John Wiley & Sons, 633p.

[50] Silva, O., Garcia, A., de Lucena, C. The Reflective Blackboard Architectural Pattern for Developing Large-Scale Multi-Agent Systems. Proceedings of the 1st International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (In Conjunction with ICSE 2002), Sunday, May 19, 2002, Orlando, Florida, USA.

[51] Subramonian, V. Gill, C. 2001. Towards a Pattern Language for Networked Embedded Software Technology Middleware. Submissions to the OOPSLA 2001 workshop 'Towards Patterns and Pattern Languages for OO Distributed Real-time and Embedded Systems', Marriott Hotel, Tampa Bay, Florida/USA, October 14th, 2001. 7 p.

[52] Sun microsystems. Designing Wireless Clients for Enterprise Applications with Java Technology. Available on-line at: http://java.sun.com/blueprints/earlyaccess/wireless/designing/designing.pdf

[53] Telemanagement Forum. On-line at: http://www.tmforum.org/

[54] Tikkala A., Matinlassi M 2002. Platform Services for Wireless Multimedia Applications: Case Studies, In: 1st International Conference on Mobile and Ubiquitous Multimedia, December 2002, Oulu, Finland

[55] Völter, M., Kircher, M., Zdun, U. 2002. Object-Oriented Remoting - Basic Infrastructure Patterns. In: Hruby, P. and Soerensen, K. (eds.) Proceeding of the First Nordic Conference on Pattern Languages of Programs, VikingPLoP 2002, Microsoft Business Solutions, pp. 201-226. Available on-line at: http://plop.dk/vikingplop/

[56] Welch, L., Marinucci, T., Masters, M., Werme, P. 2002. Dynamic Resource Management Architecture Patterns. Proceedings of the 9th Conference on Pattern Languages of Programs, PLoP 2002, Monticello, Illinois, September 8th-12th, 2002, 13 p.

[57] Yang, S., Tsai, J., Chen, I. 2002. Development of Wireless Embedded Systems Using Component Based Software. International Journal of Software Engineering and Knowledge Engineering, Vol. 12, No. 2, pp. 135-153

[58] Yuan, M., Long, J. 2002. Build database-powered mobile applications on the Java platform. Available on-line at: http://www.javaworld.com/javaworld/jw-01-2002/jw-0118-midp.html

[59] Landay, Borriello, "Design Patterns for Ubiquitous Computing", IEEE Computer, August 2003, pp 93-95