

	WISA Reference Architecture Deliverable ID: D4 (Part B)	Page : 1 of 53
		Version: 2.03 Date: 23 Oct 03 Status : Proposal Confid. : Restricted

 <p>WIRELESS INTERNET SOFTWARE ENGINEERING IST-2000-30028</p>	Title: WISA reference architecture
	Version: 2.02 Date : 23 Oct 03 Pages :
	Author(s): J. Kalaoja; E. Niemelä; A. Tikkala; P. Kallio; T. Ihme, M. Torchiano
	To: WISE CONSORTIUM
The WISE Consortium consists of: Investnet, Motorola Technology Center Italy, Sodalìa s.p.A, Sonera, Solid EMEA North, Fraunhofer IESE, Politecnico di Torino, VTT Electronics	Printed on: 15-Dec-03 10:46
Status: <input type="checkbox"/> Draft <input type="checkbox"/> To be reviewed <input checked="" type="checkbox"/> Proposal <input type="checkbox"/> Final / Released	Confidentiality: <input type="checkbox"/> Public - Intended for public use <input checked="" type="checkbox"/> Restricted - Intended for WISE consortium only <input type="checkbox"/> Confidential - Intended for individual partner only


Deliverable ID:	D4 (Part B)
Title:	WISA architectural knowledge base (WISA) and Reference Architecture (WISA/RA)
Summary / Contents:	<p>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.</p> <p>This document presents part of the WISA knowledge base for wireless service engineering. WISA/RA is the cornerstone of the WISA knowledge base. The document has been structured around the three major parts of the WISA knowledge base: 1) the taxonomy of wireless services, 2) architectural style and pattern guidelines, and 3) WISA/RA. These parts will be utilized in the development of the pilot services in the 2nd iteration phase of the Wise project. The reusable architectural assets are in Part D.</p>

TABLE OF CONTENTS

1. INTRODUCTION.....	6
2. ABBREVIATIONS.....	8
3. OVERVIEW of WISA AND WISA/RA	10
4. TAXONOMY OF WIRELESS SERVICES	13
4.1 Wireless Service architectures.....	13
4.2 Standards and classifications.....	14
4.3 Basic Taxonomy	15
4.4 End-User services	16
4.5 Application domain support services	17
4.6 Generic platform services	18
4.7 Technology platform services	19
4.8 Service management services.....	20
5. Quality attributes of WISA.....	22
5.1 Quality attributes observable at run-time	22
5.2 Quality attributes not discerned at run-time	23
5.3 The real quality of a wireless service	25
6. Guidelines for using architectural styles and patterns.....	27
6.1 The application order of patterns	27
6.1.1 Patterns for pattern-based software development.....	28
6.1.2 How to select a pattern	29
6.1.3 Representing the proper application order of patterns	31
6.2 Applying patterns to conceptual architecture.....	31
6.3 Application examples of patterns to concrete architecture.....	32


	WISA & Reference Architecture Deliverable ID: D4 B	Page : 3 of 53
		Version: 2.00
		Date: 23 Oct 03
		Status : Proposal Confid. : Restricted

6.3.1	Decomposing the software into modular parts	32
6.3.2	Decomposing the software into layers	33
6.4	Applying patterns to distributed systems and middleware	34
6.4.1	Middleware	34
6.4.2	Basic building blocks when working with RPC middleware	35
6.4.3	A pattern language for building networked systems and middleware	36
6.4.4	Examples and experiences	38
7.	WISA REFERENCE ARCHITECTURE (WISA/RA)	41
7.1	Conceptual structural view	41
7.2	Conceptual deployment view	43
7.3	Conceptual Behaviour view.....	46
7.4	conceptual development view.....	46
8.	REFERENCES.....	51

	WISA & Reference Architecture	Page : 4 of 53
	Deliverable ID: D4 B	Version: 2.00
		Date: 23 Oct 03
		Status : Proposal Confid. : Restricted

CHANGE LOG


Vers.	Date	Author	Description	Comments
2.02	Oct 7	J. Kalaoja, T. Ihme	<ul style="list-style-type: none"> VTT internal review comments updated to taxonomy Updated overview chapter partly based on Marcos proposal Moved pattern descriptions to D4d based on Marcos proposal Added conceptual deployment view models Updated Visio pictures that were corrupted Revised quality attribute chapter to be more wireless specific Added guidelines to choose patterns 	<ul style="list-style-type: none">
2.01	Aug 29	J. Kalaoja	<ul style="list-style-type: none"> Enhanced rationale and discussion of WISA taxonomy 	<ul style="list-style-type: none"> Written comments and new D4b/d division and overview chapter proposal from Marco Torchiano Review comments: Eila Niemelä, Päivi Kallio, Tuomas Ihme
2.00	Aug 19	T. Ihme M. Torchiano	<ul style="list-style-type: none"> Part of wireless patterns transferred to deliverable D4d 	<ul style="list-style-type: none">
1.09	7 Jan '03	E. Niemelä	<ul style="list-style-type: none"> Guidance for applying WISA in Pilot2 . 	<ul style="list-style-type: none"> Guidance has initially made by Patricia Lago, modified by Eila Niemelä and Päivi Kallio.
1.08	Dec 20	J. Kalaoja	<ul style="list-style-type: none"> Updated according to the comments from the reviewer. 	<ul style="list-style-type: none"> From Jürgen Münch.
1.07	Dec 05	P.Kallio J. Kalaoja	<ul style="list-style-type: none"> Added some minor modifications 	<ul style="list-style-type: none">
1.06	Dec 02	E. Niemelä	<ul style="list-style-type: none"> Updated according to the comments from the reviewer Added some parts from the journal article to TSE under work. 	<ul style="list-style-type: none"> From Maurizio Morisio
1.05	Nov 20	J. Kalaoja A. Tikkala	<ul style="list-style-type: none"> Updated Service Mgmt based on comments by Roberto Added Data Mgmt based on comments by Tapani Kilpi Updated Technology Platform Services chapter. Updated Transport Service 	<ul style="list-style-type: none"> From Roberto Tiella about Service Management and about Transport service at Nov 15. From Tapani Kilpi about Data Management at Nov 15. From Filippo Forchino about Transport Service at Nov 19. Received input for the GameUI of Pilot2 from Daniela Boggio.
1.04	Nov 19	E. Niemelä	<ul style="list-style-type: none"> Major changes to summary, introduction, WISA/RA structural and development views Minor changes to taxonomy, styles 	

	WISA & Reference Architecture		Page : 5 of 53
	Deliverable ID: D4 B		Version: 2.00
			Date: 23 Oct 03
			Status : Proposal Confid. : Restricted

		P. Kallio	<ul style="list-style-type: none"> and patterns and some generic services Minor changes to the language and style 	
1.03	Nov 08	E. Niemelä P. Kallio J. Kalaoja A. Tikkala	<ul style="list-style-type: none"> Quality attributes added Vocabulary removed HUS completed Minor changes to other generic services The overview of WISA refined Refined styles and patterns Architectural styles and patterns Taxonomy, Service Mgmt component MMS and IM/P services 	<ul style="list-style-type: none"> Input for GUI library and Game Engine requested from Daniela Boggio and Filippo Forchino Nov 12 Input for Negotiation protocol requested from Davide Brugali (Oct 30, Nov 4, Nov 8 and Nov 18). Comments from Roberto Tiella requested Nov 14 Comments from Tapani Kilpi requested Nov 14
	Oct 29	E. Niemelä	<ul style="list-style-type: none"> Updated structure of the document Introduction added Template for documenting styles, patterns and generic services 	<ul style="list-style-type: none">
1.01	Oct 28	J. Kalaoja	<ul style="list-style-type: none"> Structure of the document Service categories from D4 v.0.0x Conceptual WISA including categories Styles and patterns from D4 v. 0.0x List of services to be imported to WISA 	<ul style="list-style-type: none">

APPLICABLE DOCUMENT LIST

Ref.	Title, author, source, date, status	Identification
1	D4A - Achitectural guidelines	v. 2.00
2	D4C - Analysis of pilot architectures	v. 1.04
3	D4D - WISA handbook	v. 1.00
4	D3 - Management of heterogeneous clients	v. 2.1

	WISA & Reference Architecture Deliverable ID: D4 B	Page : 6 of 53
		Version: 2.00 Date: 23 Oct 03
		Status : Proposal Confid. : Restricted

1. INTRODUCTION

This document presents the WISA architectural knowledge base and the reference architecture of wireless Internet services. A reference architecture is an informal or formal architectural model that is community-widely accepted and reused. The WISA knowledge base includes the taxonomy of wireless services, WISA/RA reference service architecture for the development of the middleware services useful in wireless service engineering and the basic services and components that realize WISA/RA. Brief instructions to use the WISA knowledge base will also be given.

The scope of this document is to create an architectural knowledge base about reusable assets that can be utilized in wireless service engineering to develop a particular end-user service. Therefore, WISA/RA and its basic services have been documented as 3rd party components and services that are provided by component and service developers, application providers and content providers to wireless service providers that act as integrators in the development of wireless services in multi-organizational development settings.

The goal of WISA/RA is to support the developers in the design of the software architecture of a new wireless service. Therefore, WISA/RA tries to give ready-made answers to:

- What requirements should be considered In reply to the development of a new wireless service? This provides a good starting point for the service architecture development.
- What are the best practices (i.e. styles and patterns, quality attributes) for architecting a wireless service? The purpose is to encourage service developers to use the existing knowledge in order to fasten the development of a new service and increase the quality of a service, the final product.
- What support services are needed for a new wireless service? The most complicated services that take most of the development time are not visible to the end-users but prerequisites on which applications can be developed.
- What available components could be used as such and which ones need to be adapted or developed? This provides the starting point to estimate investment required for the development of the service product.

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 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 four 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 the WISA knowledge base and WISA/RA, the reference architecture of wireless services. In their development, the following approach has been applied:

- Services are categorized according to their intended use.
- Architectural styles and patterns are used as reusable assets.
- Quality attributes set to the category of services or a service have used as driving factors in selection of styles and patterns for a particular service or a set of services.
- WISA/RA has been described from three viewpoints: structural, deployment and development. The structural view is important in order to understand the place and relations of a service in a wider context. The deployment view provides alternative allocation models for service guiding end-user service

developers in the use of a service in a particular context. The development view defines the completeness of the services inserted to WISA/RA.

- Descriptions of the services are included in the WISA knowledge base. 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 has put on the quality, features and interfaces a service provides to its users, not its internal functional properties.

The WISA knowledge base and the guidelines to develop service architectures and analyze their quality issues has been developed and will be refined according to the iterative progressing plan presented in Figure 1. Until now, two iteration phases (iteration 0 and iteration 1) have been completed. This document describes the first iteration of the WISA knowledge base and WISA/RA that will evolve until the end of the Wise project.

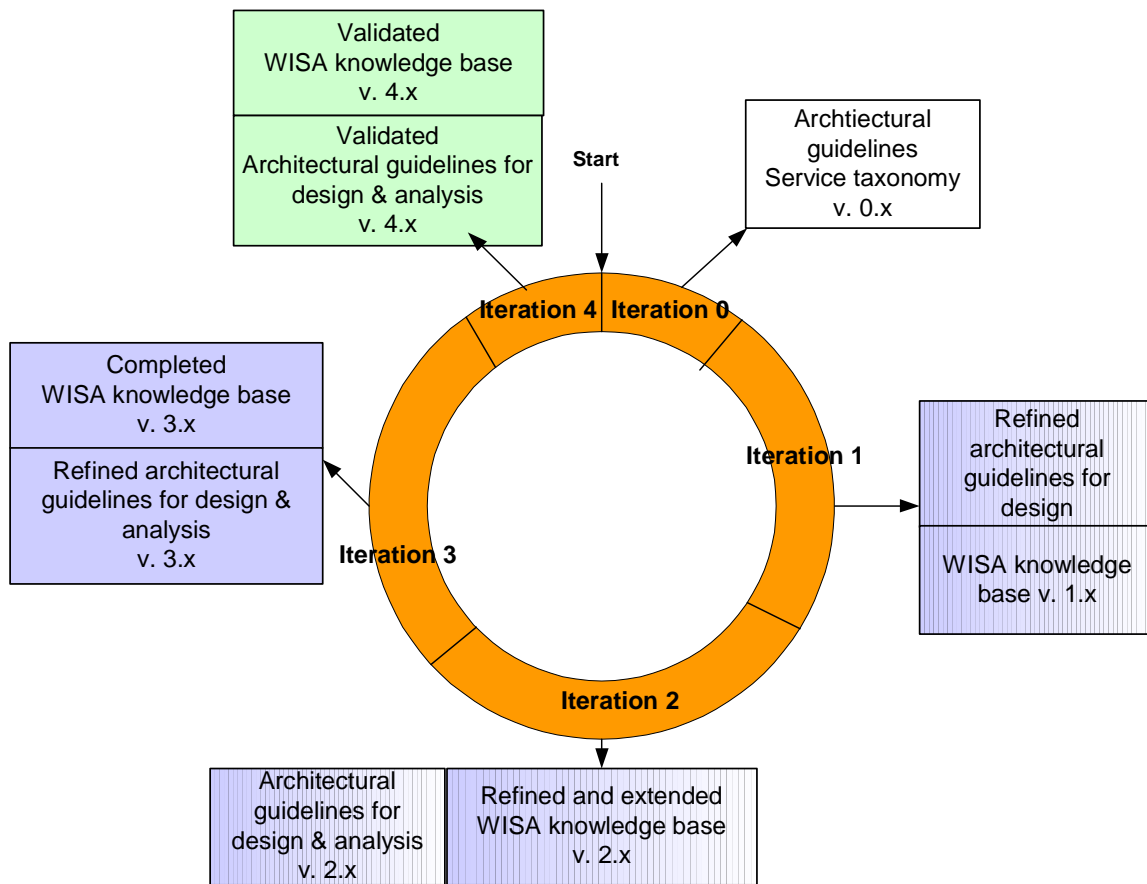




Figure 1. Iterative development of WISA architectural knowledge base.

	WISA & Reference Architecture Deliverable ID: D4 B	Page : 8 of 53
		Version: 2.00 Date: 23 Oct 03
		Status : Proposal Confid. : Restricted

2. ABBREVIATIONS

API	Application Programming Interface
ATM	Asynchronous Transfer Mode
BSS	Business Support Systems
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
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
OMA	Open Mobile Alliance
OS	Operating System
OSE	Open System Environment
OSI	Open Systems Interconnection Model
OSS	Operating Support Systems
P2P	Peer-to-Peer
PAC	Presentation-Abstraction-Control
PAAs	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

	WISA & Reference Architecture Deliverable ID: D4 B	Page : 9 of 53
		Version: 2.00 Date: 23 Oct 03
		Status : Proposal Confid. : Restricted

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. OVERVIEW OF WISA AND WISA/RA

In this section we give an overview of WISA architectural knowledge base that defines a reference architecture for wireless services (WISA/RA) and the guidelines for its use. The key idea behind the development of WISA is to use the existing knowledge, standards related to wireless services, quality attributes of wireless services, applicable architectural styles and patterns, and existing concepts, services and components that are considered as the driving forces in wireless service engineering. The corner stones of the WISA knowledge base are the following artifacts used as reusable assets in the development of wireless services:

- Taxonomy of wireless services defines service categories and their typical features to which (pilot) service developers can map their own service under development.
- Quality attributes guidelines defining their meanings and importance in service architecture development.
- Architectural style and pattern guidelines for architecting wireless services.
- Reference architecture of wireless Internet services defines conceptual and concrete architectural reference architecture views that provide understanding what enabling services are required from a platform for wireless services, how they relate to each other, how they can be used and what is their status of completeness.
- An wireless service development handbook, which contains tools for wireless service development in the form of:
 - Typical wireless architectures that the concrete architecture of a wireless service can be based on
 - Style and pattern catalog that provides descriptions of most common styles and patterns usable in development of wireless services.
 - Service catalog that provides ready-made services (developed in WISE or available from other sources) appropriate in the development of wireless services either as COTS/MOTS or as examples for architecture design.

The mapping of these elements to the documentation structure of the D4 is shown in Figure 2.

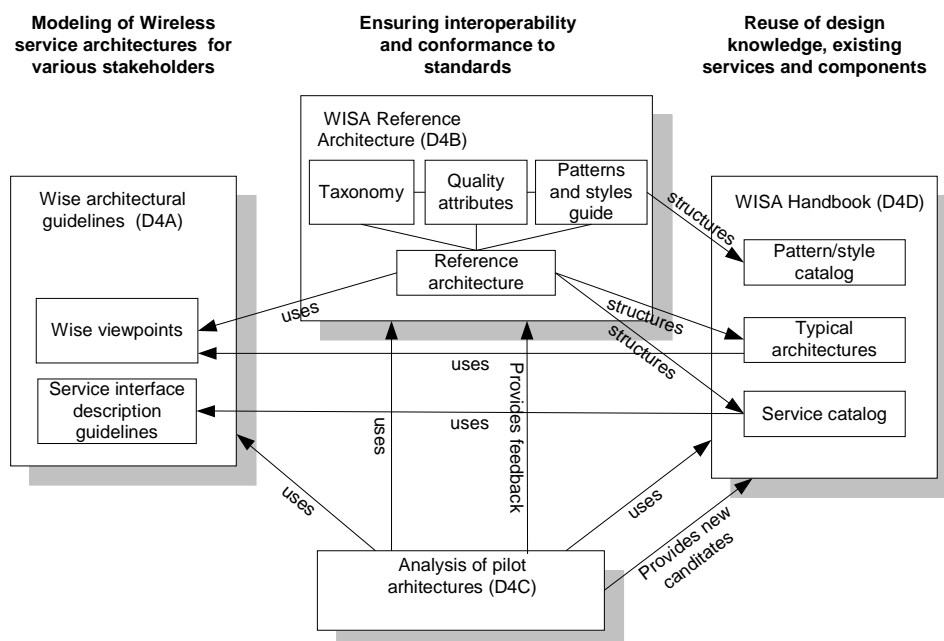


Figure 2: The roles and relations between D4 documents in Wise.

Existing standards and standardization activities form the baseline of the service taxonomy. The service taxonomy defines the main categories, called domains, and example services of these domains. Typical features further characterize services. The purpose of the service taxonomy is to guide the developers of a wireless service to map their problem to the certain domain(s) and assist in identifying the required support services and features of these services.

Quality attributes and architectural styles and patterns provide the basis for all kind of architecting. Quality of a wireless service is the most important criterion for end-user to buy, use and pay for a service. The quality of a service, however, means different things to the different kinds of stakeholders involved in the development than it means the service users (ref. 2, [19]). Meanings of quality attributes are often confused and misunderstand, and therefore, their definitions are included to the WISA knowledge base. In selecting the quality attributes of most importance for wireless services we created a framework, a quality stack, that link the business stakeholders of wireless services to the main domains of wireless services and through them to the quality attributes of these domains. This mapping between stakeholders and quality attributes assist the developers to prioritize quality attributes and concentrate on those that produces the greatest benefit not only to the service developer but also to the users of the service.

Architectural styles and patterns are tried and tested models to solve a particular kind of problem and meet the predefined quality requirements. Therefore, they are reusable assets to be included in WISA and utilized in the development of the reference architecture of wireless services, its basic services and also used in their applications, i.e. the instances of WISA/RA. This quality-driven style-oriented approach has exemplified by two generic platform services presented in [17].

The conceptual structure of WISA Reference Architecture (Figure 3) depicts the main domains of the WISA taxonomy as a layered architecture that also supports a direct interaction between nonadjacent layers if it is required by end-user applications. WISA/RA provides a conceptual architecture that can be used as a corner stone in the architecture development of a new service. This in turn assists service developers to find the services already represented in the service catalog and concentrate the development effort on new services and components needed in the new end-user service.

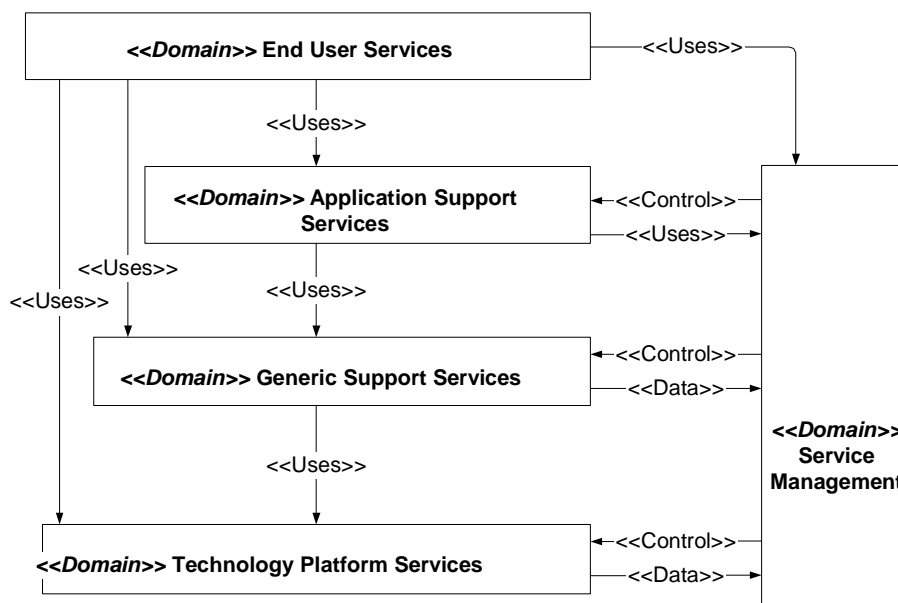



Figure 3. Overview of conceptual structural view of WISA/RA.

	WISA & Reference Architecture Deliverable ID: D4 B	Page : 12 of 53
		Version: 2.00 Date: 23 Oct 03
		Status : Proposal Confid. : Restricted

Because wireless service engineering is heavily based on the use of 3rd party components and services WISA includes a reuse repository, called WISA service catalog. The idea is to collect, adapt and document existing concepts, services and components that are valuable for wireless service engineers in the sense of use them as such, modify them to a new context or absorb some idea from them to develop a new service. The aim of WISA service catalog is to create community-wide assets that improve the quality of wireless services and speed up their development. The WISA knowledge base gives a comprehensive framework to develop the architectures of wireless services by focusing on cost effectiveness and quality that are fundamental requirements of future wireless services.

4. TAXONOMY OF WIRELESS SERVICES

This taxonomy defines a detailed classification of wireless services and their enabling services in order to gain better understanding of such services and provide basis for the structural view of WISA reference architecture and classification of services in the WISA service catalog.

First two chapters explain the rationale behind the developed taxonomy in form of analysis of what wireless services are and what kind of standards and classifications is already available. The basic idea of taxonomy is then introduced with the selected division into main domains. Each of the domains is then discussed in more detail in the remaining chapters.

4.1 WIRELESS SERVICE ARCHITECTURES

The wireless services often require the integration of services from various development organizations, and the layering of service logic is done separately by each of the organizations. This may result that a service deployed by integrating the service logic and functionality in a proprietary and service specific manner with network elements and terminals. Another problem is that the potential for finding enabling functionality, which could be shared by several services, is not fully considered. Some examples of what such shared enabling functionality are the support for location-based services or service management functions.

Service architecture is the architecture of applications and middleware i.e. the idea of service architectures is to consider not only the service as it is visible to an end-user, but also in all the other enabling services needed to make that service functionality possible. A “traditional” way of visualizing the layering of architecture of a software application is to put user interface on the top, middleware and/or operating system in the middle, and hardware on the bottom in (Figure 4 A). For simplicity the software layers used in the non-wireless UI have been left out of the layered architecture. A more service-oriented way of visualizing of software architecture is shown in Figure 4 B.

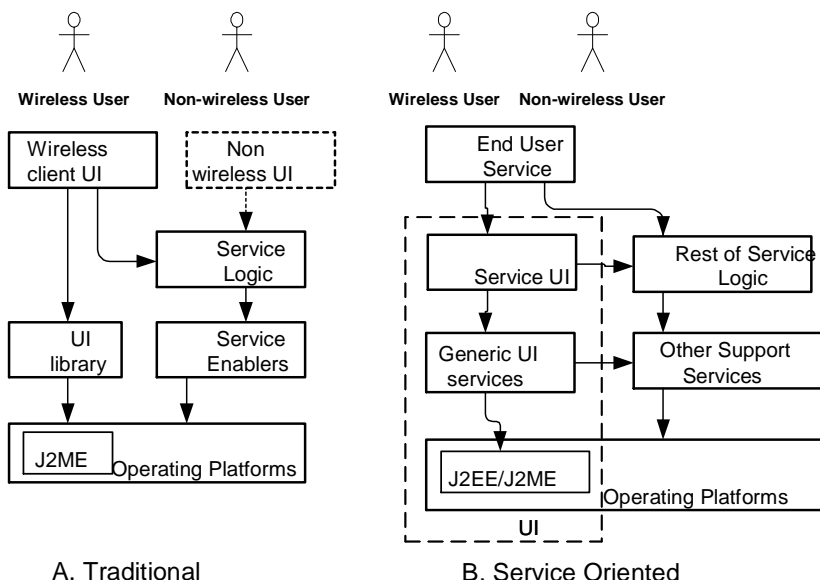



Figure 4. Visualizations of layering of wireless software architecture

The service-oriented approach promises to help the designer keep better understanding of the overall service and keep the relations between service components understandable. For example the architecture of user interface can be kept as a consistent horizontal design problem with clear interfaces to other parts of the service architecture even when service logic is divided between organizations. The relations between

	WISA & Reference Architecture Deliverable ID: D4 B	Page : 14 of 53
		Version: 2.00
		Date: 23 Oct 03
		Status : Proposal Confid. : Restricted

services should be kept logically clear with no technology dependencies. The service architecture is often build dynamically at runtime and a same support service may be involved in the architectures of several services for different or same end-user. Because of this the service interfaces may need to be customized at runtime for integrating the service into the architecture of a specific end-user service.

For example now the user interface (UI) should not be considered only as a top layer component of a service but its service architecture spans vertically into several layers. On top of the user interface is an enabling service that manages the application specific user interaction for the wireless service. This service specific UI itself can use application independent generic UI primitives. These primitives use one or more platform specific APIs provided by the various wireless devices.

The emergence of service-oriented architectures has much to thank for the IP based networking, where it is easy to create direct IP based communication links between services, and integrate technology platforms they require, using separate distribution nodes. The client side is quite standardized because of the use of browser technologies. A main benefit of service-oriented thinking in development of wireless services could be that this kind of flexibility helps to integrate services by different business stakeholders with different technologies into single service. However, for the wireless services the limited capacity of wireless links may make it more difficult to deploy services freely into separate distribution nodes. The display, keyboard, number and capacity of IP based communication links in wireless devices are limited. A local proxy may have to be deployed into wireless terminal for data intensive service in order to keep the communication need over the wireless link at minimum. This means that the various wireless device operating environments have to be supported by the support service and sometimes there may not be all the device resources left that the service requires. Agent based platforms and dynamic customization combined with service orientation may provide one solution for these problems.

4.2 STANDARDS AND CLASSIFICATIONS

The standardization work in this field is ongoing task, and to keep in par with the current state of the art requires continuous effort of tracking the status of work in the forums related to standard development. This chapter lists some of the most important sources of information used as the basis of this taxonomy.

The UMTS forum [11] has given a division of the end-user application domain services available for third generation mobile services. Compared to the architecture-oriented view adopted in this taxonomy the interest lies in the business- and mobile technology oriented aspects of services.

TM Forum's enhanced Telecom Operations Map (eTOM) process model includes three vertical domains associated with the lifecycle management process functions of infrastructure, product and supply chain and several horizontal layers needed for service management processes [12]. Of those the vertical domains for service lifecycle management were considered relevant for structuring the service management services domain of this taxonomy.

Open Mobile Alliance (OMA) has identified at least two classes of support services: service enablers that are used by a wireless service directly and common enablers that can also be used by wireless service enablers. Several classes of enablers have been identified, which can be into various levels of this taxonomy. Nokia Mobile Internet Technical Architecture (MITA) is largely based on this work and introduces a common platform architecture for developing future mobile internet services, which may guide the work on defining concrete architectural views for WISA.

Open Systems Environment (OSE) [18] provides classification of generic middleware services. The classification has been adopted by the taxonomy as the preliminary basis of classification of generic platform services.

Schmidt [31] decomposes middleware into multiple layers and describes R&D efforts related to each layer. Those layers can also be considered as a structure for commercial middleware components. The idea is quite similar to the main domain division adopted by this taxonomy although this article was not available at the time of choosing this division (v0.1 of D4). These middleware layers are described in more detail in the context of applying patterns for middleware (Chapter 6.3.2.1), and may be most useful in designing a concrete architecture of a wireless service or a platform for such services.

The user equipment profiles (UEProfile) defined for mobile terminals to be used by WAP services have been as the basis of classifying technology platforms.

Relevance to be considered in next iteration:

- Intelligent networks as applied to wireless services (WIN)
- Semantic Web and ontologies

4.3 BASIC TAXONOMY

A main requirement for this taxonomy was that it should help to understand a wireless service from service-oriented perspective. Another requirement was to help identify the potential for both horizontal and vertical reuse of service logic and functionality and to help to define commonly usable open interfaces for the enabling services. To achieve these goals, the taxonomy uses two dimensions for categorizing the wireless services:

- The first dimension of taxonomy classifies the services based on how directly the service or enabling service is related to the end-user mission for which he is using the service.
- The second dimension of taxonomy classifies the services into categories of similar mission related functionality provided by the service.

The potential for horizontal reuse is related to the second dimension of taxonomy and potential for vertical reuse to the first dimension of the taxonomy.

Based on the first dimension of taxonomy the constituents of wireless services have been divided into five domains. The domains are described in Table 1. The rationale for selecting this division is discussed separately in the following chapters.

Table 1. Domains of wireless services.

Domain	Description
End-user services	Services visible as applications to end-users. A wireless service can have both users using wireless terminals and users using non-wireless services (e.g. operators of wireless emergency service). Users of service management systems are left out of the scope of this taxonomy and not considered here as end-users.
Application domain support services	Services that provide generic services for a specific application domain on which end-user applications rely, but that are not usually targeted for the end-user as a service (e.g. a game engine). Application domain can be scoped for example by specific knowledge required, or similar business logic.
Generic platform services	Services that are needed by the end-user services and application support services, but are more generic and not directly related to any application domain (e.g. GPS location service).
Technology platform services	Services related to specific implementation technology (software or hardware) choices in either mobile terminal or server side (e.g. WML, Web browser, mobile phone, Java).
Service management	Services that are needed to make the service available and link it to

	business processes. As such they usually are not directly related to the purpose of end-user applications (e.g. service downloading).
--	---------------------------------------------------------------------------------------------------------------------------------------

Flexibility is an important requirement for wireless service architectures. The need for flexibility arises from the diversity of underlying platform technologies for networking, wireless terminals that must be supported, various business models related to service revenue sharing, and different environments in which the services are used. The flexibility can be achieved by providing variation with optional service implementations, or customization of the services by configuration and adaptation. The requirements for the service options and customization are analyzed in this taxonomy by identifying the typical features in each of the categories and domains.

The features identify the variation points for the wireless service architecture that the customization and adaptation can be based on. For a single service these variation points may be needed in several levels of the taxonomy. When adapting the service architecture the service visible to end-user may not only need to customize the enabling services directly used by it, but it may also need to consider the features of all domains presented here. An obvious example of this is the adaptation of a wireless service and its enabling services to the platform provided by the particular end-user wireless terminal. Another type of adaptation needed is so called context-awareness. This means that a wireless service knows something about the environment or locale in which the service user is in, and can provide a customized service based on that information. The context information is not shown directly in this taxonomy, but the variability needed from service and its enabling services could be reflected into features discussed in various levels of this taxonomy.

4.4 END-USER SERVICES

The success of a wireless service depends on its benefit and quality to the end-user compared to the cost of the service. Of those the quality is an important issue for the service architecture. Ensuring the effect of required end-user service quality on the service architecture is not straightforward. For example: although a mobile gaming service requires high performance for seamless operation, it should not get more communication bandwidth in expense of an emergency or telemedicine service. Alternatively, a service can be valuable or even essential for the end-user but requires only small amount of network resources. This means that the business models for the wireless services cannot be reflected in the service architecture simply as revenue created by the network traffic, or use of various resources. These problems will become more difficult when developing generic enabling services or wireless middleware platforms.

One of the main purposes of the end-user service categories proposed here is to help to identify and analyze how the tradeoff between different architectural options such as selected priority of quality attributes and choice of architectural styles for service architecture should be considered. To help this, some of the most typical characteristics of end-user services that have an effect on service architecture were identified as features (Table 2). The division of end-user service categories is now made so that a typical choice of these features characterizes the category. The architectural quality attributes associated to service architectures are discussed in more detail in chapter 5, and architectural styles and patterns in chapter 6.

Table 2. General characteristics of wireless end-user services.

Feature group	Optional features
Communication style	Single user / Multi-user Networked / single terminal Server-centered/ Peer-to-Peer One-to-one / Multicast / Broadcast
Usage style	Interactive / Non-interactive Content Push / Pull

Resource requirements	Customized / Generic service Persistent / Volatile (i.e. single use) Communication rich / light Priority/relative importance (High/Normal/ Low)
Quality of Service	Single channel / Multichannel Security (High/Normal) Performance (High/Normal/Low) Availability (High/Normal)

The selected end-user service categories are presented in Table 3. This set of categories is in no means complete but should help to identify a preliminary set of quality attributes required from the service architecture. The naming of the categories is sometimes difficult and is based here on the most typical end-user services in the category. Sometimes the division between categories is not apparent. If services had been categorized from business viewpoint the division would have been quite different. For example teleworking & B2B communication could be a category of own, but they are not separated here because it is difficult to differentiate in practice the communication needed for business from communication for personal life. If needed, some categories like mobile entertainment could be divided further. Often an end-user service is a composite of services belonging to different categories. For example a mobile game can provide a possibility to purchase new options for the game. In that case that sub-service could be considered in the mobile commerce category.

Table 3. End-user service categories.

Category	Example applications	Typical features
Mobile Entertainment (Nonessential but resource demanding)	Mobile games, Mobile Music/Video	Communication Rich, High Performance, Low Priority, Multicast, Multichannel
Mobile Information (Service provider originated information)	News, Weather, Travel info, Yellow pages	Light, Low Performance, Broadcast
Mobile Communications (Peer to peer communication)	Rich call, Messaging, Email, Teleworking, Virtual Home Environment, Telemetry / Monitoring	Secure, High Availability, Multichannel
Mobile Commerce (Service provider originated and financially critical services)	Mobile Shopping, Mobile Banking, Stock Exchange	Secure, light, High Availability, High Performance, Multicast
Critical end-user services (possibility for hazard for human life)	Telemedicine: X-ray image exchange, Patient Monitoring, Fire alarms, Emergency (911/112), e-/m-Health services	High Security, High Availability, High Performance, High Priority

4.5 APPLICATION DOMAIN SUPPORT SERVICES

It is important that the development of a new wireless service is cost effective when compared to the anticipated income produced by the service. Therefore development of a service from scratch should be avoided unless the application domain of service is completely new. The application domain support services could provide the building blocks from which similar services can be easily constructed with

minimal development cost. An end-user application might typically uses services of one or two of these categories. The proposed division of application domain support services is based on the common types of operations and information needed in the particular category. The categories in this domain are continually evolving and completely new categories arising.

Table 4. Application domain support service categories.

Category	Example services	Example features
Wireless Gaming Support	Mobile game engines, High score lists, Player Community Services, Game / Demo downloading	Game category (action/adventure...)
Wireless Multimedia Support	Music Download/Streaming, Video Download/Streaming, Electronic Postcards	Bitrate of stream
Communication Support	Email, SMS, EMS, MMS Services Mobile Chat Board Service	Multimedia enabled
Information Content Sources	News content servers: Entertainment, Sports, Business, Weather forecasts Address, phonebooks, service catalogues, Advertisement	Push/Pull, delay of information, detail level
Location-Based Services	Positioning, GIS: Map	
Mobile Shopping	Shopping Basket, Secure paying	Credit cards supported
...New and emerging categories	...Pioneering services...	...Proprietary features...

The features in this domain vary greatly depending on the category. Definition of services in some categories, e.g. the enablers for location based services, are governed by standardization bodies. More often however, the features of a successful pioneering service become an ad-hoc standard for the particular category. In the non-wireless services one of such areas is the game development. The evolution of enabling technologies for gaming has been so fast that a single standard gaming platform has not been dominant. However, the sub-categories of games, basic features of development and runtime platforms (i.e. game engines, and content support tools) are well known. Usually a game engine with small enhancements has been the basis of at least several new games and licenses of it sold to other game developers.

4.6 GENERIC PLATFORM SERVICES

Obtaining technology and platform independence is especially important for wireless services because of the variability of both wireless terminals and server platforms of network operators. One problem is the integration of legacy systems with different architectural quality attributes. For example in the development of emergency services like e911, the high security government networks has to be integrated with the infrastructure of commercial network operators.

Generic platform services should provide technology independent generic services that are not specific to a single end-user application domain. The division of generic support services here is based on the OSE categories [18] because most of these services are the kind of services typically provided by traditional middleware, and reuse of the traditional classification here is justified. The individual middleware platforms are not included in this domain, rather the common services typically provided by them. The wireless domain is not usually reflected here as distinct services, but rather as features like transparency of communication services to mobility and distribution. The services in this domain are usually defined by standardization bodies, who try to define vendor independent standard API:s. Use of these more generic enabling services rather than technology platform services directly enhances the technology and vendor independence of service architecture. The technology platform vendors advertise standard and open

interfaces provided but naturally complete vendor independence of a customer is not in their interest. Also there are several competing standardization bodies that define conceptually similar but technically different standards. Because of this the emergence of generic services in this domain as off-the-shelf is not probable but these categories and services could provide a common vocabulary with which to analyze the architecture of a wireless service in a truly platform independent way.

Table 5. Categories of generic platform services.

Category (OSE)	Example services	Example Features
Human Interaction	Support for heterogeneous user interfaces	Graphical / Textual
Workflow/Task services	Service Deployment Support Service Activation / Configuration Support Service Chaining Support	Static/Dynamic Static/Dynamic Static/Configurable
Processing services	Time of day Compression / Decompression algorithms: MPEG etc.	Protocol versions
Data Management services	Database management Large data object transfer Replication of data	ODBC, SQL etc. support Uploadable/Downloadable, Size of data, Content Types Replication strategies
Resource management services	Resource reservation Resource modification Resource monitoring	Priorities, mutex Static/Dynamic
Communication services	Messaging mechanisms Name services Brokers	Streaming / packet based Local / global Transparency: Location, Name etc.

4.7 TECHNOLOGY PLATFORM SERVICES

Platform independence is a generally accepted goal in software engineering. Problem often is to understand what is meant by a platform. The goal of technology platform services domain in this taxonomy is to identify the types of platforms used in wireless services. Platforms are usually more or less interoperable technology choices providing services on various levels of this taxonomy, but can be also understood as services themselves. Platforms are not bad idea - in fact platforms can provide an excellent base for fast and efficient development of new wireless services. The inherent problem of platforms is the interoperability (or lack of it) with other platforms. The problem in wireless services is the rapid evolution of platforms on both the terminal and server side. Only recently it has been identified that a successful platform for wireless services should cover both of these sides in order to be usable. This taxonomy as a whole can be seen as an effort to identify the kind of services a platform for wireless services should provide.

The division of categories for technology platform service in this taxonomy is based on the types of resources the platform provides. This was choice is based partly on intuition and was considered helpful in analyzing the interoperability of specific service with another i.e. of what kind of platform dependencies have be considered for the wireless service architecture. More concretely the division is based on what kind of information is needed in order to make a service to adapt to a specific platform. The user equipment profiles (UEProfile) defined for mobile terminals to be used by WAP services are a step to this direction, and it is not by accident that the division here is similar than used in them. Conceptually, more abstract user context

information could also be discussed at this level, but in this taxonomy only the technology platforms are considered.

Table 6. Technology platform service categories.

Category	Example Services	Example features
Hardware Environment	Mobile device, Server, PDA Computing capability Hardware accelerators Communication interfaces: 2G, 2.5G, 3G, IR, Bluetooth Screen	Vendor, Model Available application memory, Computing speed Compression technologies supported Protocol versions supported Screen size, Color capable
Software Environment	Operating System: MS Windows, Symbian, Linux Java VM: J2ME, J2EE Web Server Web Browser WAP Server WAP Browser	Processes, threads, priorities, scheduling policies Packages supported Java Server Pages, Server Side Javascript Javascript support, Images, Frames, Flash player WAP version
Networking Environment	Application level protocols: TCP, UDP, HTTP Multimedia streaming	Connections Between applications/ Terminal to server, Between terminals Latencies Application to server Application to another application Frames per second, Picture Quality

4.8 SERVICE MANAGEMENT SERVICES

As it was noted earlier, the wireless service architecture should provide flexibility in order to support various business models for revenue sharing. For example a service user accepting commercials could benefit from it with higher bandwidth or lower cost. Usually each network operator has its own system for handling the wireless service customers, collecting billing information from service provisioning and its resource use, making development of operator independent services difficult. A wireless service may have a focused target group so that the fast deployment of new services into international markets is needed to get back the development costs. Currently for example international billing of service use between networks of even single mobile operator is not possible in some countries.

Because the integration with business models is a complicated issue, a service developer should not be burdened with details of it if possible. This means that the service management services, although similar to the generic support services, have quite a different role in wireless service development. The service management services are better understood as a part the telecom operations related to telecom business processes. Table 7 shows horizontal eTOM [16] layers and examples of operations in each layer. Usually these (horizontal) operations are important to for example service management personnel of network operators, and should be transparent to the wireless service and its developer. Because of this these layers are left out of this taxonomy but understanding of them helps to understand how the wireless services link to business processes.

Table 7. Telecom Operations Related to Service Management.

eTOM Layer	Operations
Customer Relationship Management (CRM)	Order Handling Problem Handling Customer QoS / SLA Mgmt Billing & Collections Management
Service Mgmt and Operations	Service Configuration & Activation Service Problem Mgmt Service Quality Mgmt Service & Specific Instance Rating
Resource Mgmt & Operations	Resource Provisioning & Allocation to Service instance Resource Problem Mgmt Resource Restoration
Supplier/Partner Relationship Mgmt.	S/P Buying, S/P Purchase Order Mgmt, S/P Problem Mgmt, S/P Performance Mgmt, S/P Settlements & Billing mgmt

The vertical eTOM layers seem to be more important for service developer and provide the basis for the service categories of service management services. Table 8 presents the selected categories based on these layers, and examples of support services belonging to each of the categories. These services are not usually directly related to the mission of service for the end-user, but the service or its enablers will eventually need some of these operations. For easier development of a new service, these services are often provided in a more domain specific form by application domain support services. For example a gaming support service can provide game specific operations or complete transparency to the service subscription or authentication. Note that sometimes the naming of services can be misleading, the location services in the service management domain provides location based service discovery service rather than location information for the wireless service. The profile information is handled by service management services because this information is related to the customer relationship of the end-user with the service provider.

Table 8. Service Management Service Categories.

Category	Support Service Examples	Example Features
End-user Fulfillment Support (i.e. service provisioning)	Finders, Location services Service Subscription Client Code Deployment Configuration Support	Service customization Supported Mobile Platforms During provisioning , start-up, runtime
Service Assurance Support (i.e. end user quality management)	Authentication & Authorization Context Information User Profile Mgmt. QoS/ SLA mgmt	Authorization protocols UE Profile, QoS, Server Profile, ambient Service specific, Context specific
Service Billing Support (i.e. business view)	Accounting Rating, Mediation	Billing Strategies

The features presented here are mostly related to the business models of wireless services: either for interoperability of service management platforms of various business stakeholders, their revenue sharing or resource allocation for the service based on business model.

5. QUALITY ATTRIBUTES OF WISA

This chapter introduces the most important general quality attributes of wireless software architecture. Generally, the quality of a software system is divided in two categories:

- The set of quality attributes that is observable at run-time, such as performance, functionality and usability.
- The set of quality attributes that cannot be discerned at the run-time such as reusability or integrability [4].

The main quality attributes meanings of which in the development of software systems and software services are further explained in the first two chapters. The third chapter introduces the concept of quality stack, puts emphasis on the importance of real quality of wireless service for the various stakeholders wireless services.


The quality requirements of different categories of end-user services were discussed also in chapter 4.4 in context of WISA taxonomy.

5.1 QUALITY ATTRIBUTES OBSERVABLE AT RUN-TIME

Table 9 lists the main run-time quality attributes of wireless systems. The importance of each of these in wireless service architectures is then further discussed.

Table 9. Quality attributes observable at run-time.

Quality attribute	Description
Performance	Responsiveness of the system, which means the time, required responding to stimuli (events) or the number of events processed in some interval of time. Another form of expression for performance is the number of transactions per unit time or the amount of time it takes a transaction with the system to complete
Security	A measure of the system's ability to resist unauthorized attempts at usage and denial of service while still providing its service to legitimate users.
Availability	Availability measures the proportion of time the system is up and running. <i>Mobility</i> : Is the service provided by the system continuously available when the user is no longer stationary?
Reliability	The ability of the system or component to keep operating over the time or to perform its required functions under stated conditions for a specified period of time.
Flexibility	The ability to change system structure or functionality at runtime. <i>Adaptability</i> : The service can adapt itself to its usage context <i>Variability</i> : The service can be configured to its usage context.
Usability	Can be broken down into the following areas: <i>Learnability</i> : How quick and easy is it for a user to learn to use the system's interface? <i>Efficiency</i> : Does the system respond with appropriate speed to a user's requests? <i>Memorability</i> : Can the user remember how the system operations are done between different use-times of the system? <i>Error avoidance</i> : Does the system anticipate and prevent common user errors? <i>Error handling</i> : Does the system help the user recover from error? <i>Satisfaction</i> : Does the system make the user's job easy?

	WISA & Reference Architecture Deliverable ID: D4 B	Page : 23 of 53
		Version: 2.00 Date: 23 Oct 03
		Status : Proposal Confid. : Restricted

In the case of distributed systems, **performance** is a function of how much communication and interaction there is between the components of the system. If all the components of the architecture are on the same processor then performance is a function of the amount of interaction by a subroutine invocation or it is referring to process synchronization. Performance analysis looks at the arrival rates and distributions of service request, processing times, latency and queue size if applicable. For the wireless systems especially the latencies of wireless communication channels and limited processing power of wireless terminals makes performance often the most important concern during the development. Much interest should put on optimization of communication protocols but even more on efficient division of responsibilities of architectural components (e.g. clients and servers) to minimize communication and divide processor load. The performance of server and network side is visible as the "scalability" (not to be mixed with a quality attribute) of wireless service when increasing number of users. Performance has also a great effect on the usability and satisfaction of a wireless service for the users, and failure of many early mobile services was based on the lack of performance.

Types of **security** threats could be denial of service (by flooding the target with connection requests or queries) related to availability, or *IP source address spoofing* (by assuming the identity of a host trusted by the target). Strategies against prevention, detection and response to an attack are authentication server, network monitors, and "firewall", a system constructed on top of a trusted kernel that provides security services. These are important on the fixed network (server) side or in wireless systems based on WLAN or comparable technologies. The communication standards and mobile terminal platforms govern the security in mobile terminal at the moment but in future the security characteristics of even these will become more similar to fixed terminals. The security is especially important in the category of critical services (e.g. e911, medical applications etc.) but lack of it can not be excused in any of the categories. Because of this achieving security should be transparent to a wireless service developer and inherently build into the service architecture.

Availability related to system failures can be defined as $mean_time_to_failure / (mean_time_to_failure + mean_time_to_repair)$. Designing components that are easy to modify and designing a component interaction scheme that helps to identify misbehaving culprits lowers *Mean_time_to_repair*. The availability of critical services is especially important for the end-users but it is in no means unimportant in any of the categories because of possible loss of revenue for the business stakeholders. **Mobility** of wireless service can be defined as the availability of the service when the terminal is moving, especially between different wireless network cells. The mobility is often still more an added value for the user of a wireless service than a required quality. For service developer the mobility should be transparent and handled by the generic technology platforms. However the mobility can indirectly affect performance because of varying capacity and latencies (Quality of Service) of communication channels. The seamless hand-over of service when moving out/into range of different wireless network technologies is still an issue to be solved and the required availability has to be ensured explicitly.

Reliability equals to *mean_time_to_failure*. *Mean_time_to_failure* is lengthened by making the architecture fault tolerant, by the replication of critical processing elements; by fielding a less error-prone system, which is addressed architecturally by a careful separation of concerns, which leads to better integrability and testability.

5.2 QUALITY ATTRIBUTES NOT DISCERNED AT RUN-TIME

Table 10 lists the main quality attributes of wireless services not discerned at run-time. These quality attributes are also called **non-functional quality requirements (NFR)**. Functionality is the ability of the system to do the work for which it was intended. It is orthogonal to structure, meaning that it is largely non-architectural in nature.

The interest of a non-functional quality attribute for the architecture is how it interacts with, and constrains, the achievement of other quality attributes. Non-functional qualities of a software system have great impact

on its development and maintenance, its general operability and its use of computer resources. The larger and more complex a software system is and the longer its lifetime, the more important its non-functional characteristics become.

Some quality attributes, like **reusability** and **modifiability**, have similar architectural techniques for their achievement. A similar overall purpose may also be achieved by multiple different quality properties, like portability and interoperability. Interdependencies and tradeoffs also exist between quality attributes. In case of conflict, an ordering priority between NFRs should be specified, or a preference of one NFR against another should be defined.


Table 10. Quality attributes of service architectures not discerned at run-time.

Quality attribute	Description
Modifiability	The ability to make changes quickly and cost-effectively.
Maintainability	The ease with which a software system or component can be modified to correct faults, improve performance, or other attributes, or adapt to a changed environment.
Flexibility	The ease with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed.
Scalability	The ease with which a system or component can be modified to fit the problem area.
Portability	The ability of the system to run under different computing systems: hardware, software or combination of the two.
Reusability	Reusability means designing a system so that the system's structure or some of its components can be reused again in future applications.
Integrability	The ability to make the separately developed components of the system work correctly together.
Interoperability	A special case of integrability that measures the ability of a group of parts (constituting a system) to exchange information and use the one exchanged.
Testability	The ease with which software can be made to demonstrate its faults (typically execution based) testing.

Modifications to a system can be divided in several categories of abilities. We can distinguish the ability to acquire new features, simplify the functionality of an existent system, adapt to new operating environments, or restructure system services. A restructuring ability of the system leads to a decomposition of the system into modules, which in this way encourages the creation of reusable components. **Maintainability** is the same as **modifiability**, from the architectural point of view, but a fine distinction between the two terms has to do with the type of change being installed.

Reusability is a synonym to **integrability** if the system has been structured so that its components could be chosen from previously built products. It could be seen as a special case of modifiability. Integrability measures the ability of parts of a system to work together. It depends on the external complexity of the components, their interaction mechanisms and protocols, and the degree to which responsibilities have been cleanly partitioned.

Flexibility is important in the context of the reference architecture development. **Variability** is essential for building a flexible architecture. It is possible to anticipate at least some common and variable aspects in requirements of different service members of a domain and to construct a service in such a way that it facilitates this type of variability. For this purpose the variability will be considered for the services in the wireless service catalog.

	WISA & Reference Architecture Deliverable ID: D4 B	Page : 25 of 53
		Version: 2.00 Date: 23 Oct 03
		Status : Proposal Confid. : Restricted

Testability refers to the probability that, assuming that the software does have at least one fault, the software will fail on its next test execution. It is related to the concepts of observability, to observe its outputs, and controllability, to control each component's internal state and inputs.

5.3 THE REAL QUALITY OF A WIRELESS SERVICE

The main business stakeholders, i.e. a service user, a service developer and a service provider, were used as a starting point to identify the essential quality attributes of wireless services and maps them to service architecture. Figure 5 represents an overview of the quality 'stack' that classifies qualities into **internal and external qualities** of four categories. [19]

The internal qualities are the non-functional properties of software service that are important for the developers of that part of the software in question but may be invisible or unimportant to the other stakeholders involved in the service development. The external qualities are the quality requirements that have to be visible to the stakeholders that use the software when they develop or provision the final product, a software service.

Various stakeholders in wireless services, i.e. users, application developers, platform (i.e. middleware) service developers and network operators, prefer different qualities. External quality provided by a stakeholder is a prerequisite for internal quality of another stakeholder in the stakeholder 'stack'. The real quality of a service, i.e. how well the service meets all end-user's requirements (cost vs. benefits), defines the **real added value** for an end-user. This quality is achieved only if prerequisite technical and economic qualities are met.

Applicability, i.e. how easily the application can be applied in different contexts, is a quality that the application developers are most interested in. This quality is visible as external qualities through GUI's usability, performance of the application and ease of service use by a scaling number of end-users.

Interoperability of platform services is the criterion a service developer considers a required quality of the software when a service is provisioned. Interoperability is achieved if platform services are generic and new platform services can be easily integrated by aggregating the old ones (horizontal integration). The same platform services may be usable in other applications (vertical integration) and application developers should be able to use them easily through simple interfaces. The platform services should also be portable, modifiable for different applications, expandable, maintainable and easily used and accessed by application developers.

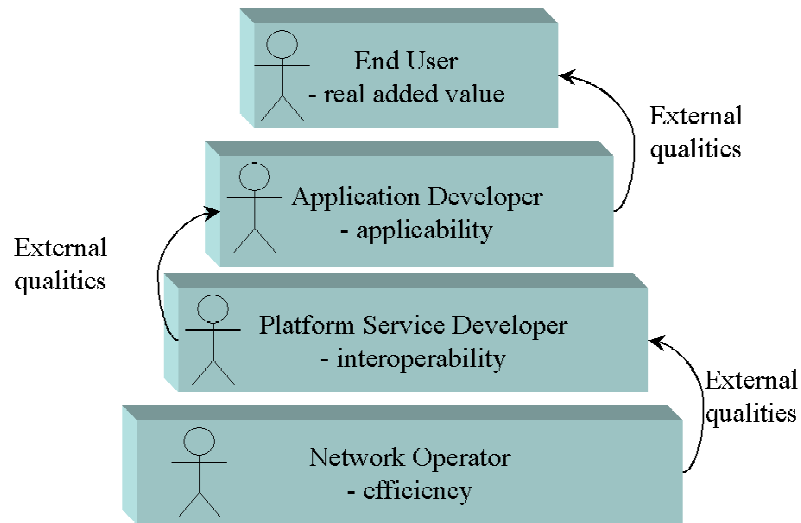



Figure 5. The Real Quality accumulates in cooperation with various stakeholders.

In summary, the driving forces behind the development of WISA/RA are portability, integrability, reusability and modifiability (see ref. 2). These key forces should be met at the same time with the other architectural drivers of service architectures, such as extendibility, maintainability, performance and simplicity.

	WISA & Reference Architecture Deliverable ID: D4 B	Page : 27 of 53
		Version: 2.00 Date: 23 Oct 03
		Status : Proposal Confid. : Restricted

6. GUIDELINES FOR USING ARCHITECTURAL STYLES AND PATTERNS

Architectural patterns [2] describe the expertise of experienced developers on fundamental overall structuring principles of software systems. Design patterns [24] describe the expertise of the developers on commonly recurring structures of communicating software components that solve general design problems within particular contexts. Architectural patterns help to design coarse-grained architectures, design patterns can be used during the whole design phase. Architectural and design patterns help to construct software architecture with specific quality attributes.

Software architects elaborate software structures, styles and patterns using the following three categories of element types: design and implementation units, runtime execution units or their abstract counterparts, and non-software elements in software's environment [3]. Architectural styles and patterns can be classified according to the three perspectives:


- styles in the module category document structures of design and implementation units,
- styles in the runtime structure category document structures of execution units, and
- styles the allocation category document the mapping from software elements to environmental structures.

Clements et al. [3] identify and discuss four styles in the module category (decomposition, uses, generalization and layered), six styles in the runtime structure category (pipe-and-filter, shared-data, publish-subscribe, client-server, peer-to-peer, and communicating-processes), and three styles in the allocation category (deployment, implementation, and work assignment). The three categories can be considered as fundamental architectural perspectives. The component-and-connector style can be considered as a super-style for the styles in the runtime structure category when the term component refers to a runtime entity. Clements et al. [3] also describe key properties of the styles such as purpose, elements, relations, computational model, topology and notations. For example, achievement of quality attributes such as modifiability, reliability, and performance, as well as build-versus-buy decisions and product line implementation, are important architectural drivers for the selection and use of the decomposition style.

Many architectural styles and patterns are combinations of several elementary patterns and styles combinations and thus may integrate software element types of different categories in one view. However, it is difficult to understand the purpose of a style (or pattern) if the types of software elements in the style remain unclear from the three fundamental perspectives. A style or pattern may be a specialization of a more general style or pattern. For example, the Tiered style [21] is a specialization of the decomposition style.

Numerous categories have been suggested for classifying software patterns, with some of the most common patterns being [2],[24],[25]: Analysis, Architectural, Design, Creational, Structural, and Behavioral, and Idioms. Sets of closely related patterns have been depicted as pattern languages. While this deliverable focuses on architectural styles and patterns, it also addresses styles and patterns in other categories when they help to solve architectural problems of wireless systems. While only few patterns describe the architectural expertise of experienced wireless and mobile systems developers, this deliverable also discusses examples and experiences in the use of trusted general purpose styles, patterns and pattern languages in architectural design of wireless systems. The deliverable addresses considerably architectural patterns for building networked systems because distribution is an important system property in all wireless systems. Schmidt et al. [25] estimate that more architectural patterns for wireless and mobile systems will emerge in the near future.

6.1 THE APPLICATION ORDER OF PATTERNS

	WISA & Reference Architecture Deliverable ID: D4 B	Page : 28 of 53
		Version: 2.00 Date: 23 Oct 03
		Status : Proposal Confid. : Restricted

The application of architectural styles and patterns to building real-world wireless systems is not a mechanical task. Many inter-related and competing requirements and design problems must be resolved and balanced when developing wireless systems. Some architectural patterns are used to define the overall software architecture for a whole wireless system and thus their proper ordering is not only hard, but also very important.

The pattern-specific guidelines of pattern descriptions are insufficient for applying patterns in building large-scale and complex real-world software architectures [13], [25]. They pay attention to some specific aspects in a particular context and ignore many other important characteristics of a system. They describe each pattern in a self-contained manner independently of many essential aspects of the whole software architecture of a system. It is hard to extract relationships between patterns and the proper application order of them from their descriptions. The pattern descriptions do not address the integration of patterns into a partial design, the combination of patterns to larger design structures, the application order of a given set of patterns, and the resolution of problems that cannot be solved by a single pattern in isolation [13].

6.1.1 Patterns for pattern-based software development

Buschmann [13] presents 11 patterns that help designers to integrate and combine architectural patterns into a partial software design in a proper application order. The patterns form a pattern language and Figure 6 summarizes the intent of each pattern and briefly illustrates how the patterns build upon each other. If a pattern uses another pattern, it points to it with a dependency arrow with the "use" stereotype. The name of the arrow indicates the purpose of the use. Due to space limitations, only the name of one of the incoming arrows of each pattern is shown in Figure 6. The shaded Piecemeal Growth pattern indicates the starting points of the pattern language.

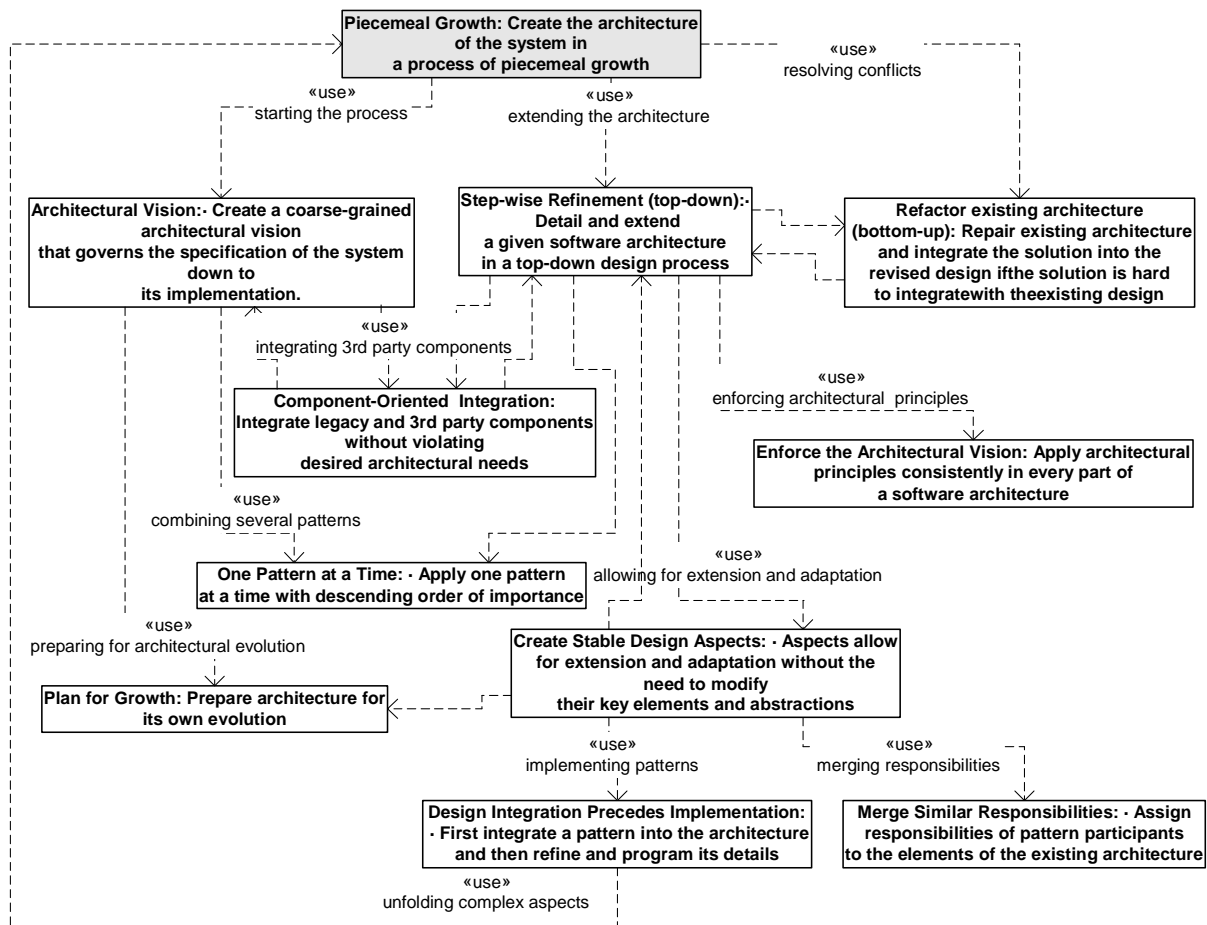



Figure 6. A pattern language that helps designers to integrate and combine architectural patterns into a partial software design in a proper application order (redesigned from [13]).

6.1.2 How to select a pattern

Figure 6 involves an iterative application of patterns from high-level into low-level patterns. Architectural patterns address coarse-grained problems in software architecture. The patterns often specify which other patterns can be used to resolve sub-problems of the original problem and complete these high-level patterns. Low-level patterns such as design patterns are often used to complete further this problem solving sequence.

Patterns are often selected and applied individually, each helping to resolve a particular architectural or design problem. However, it is often hard to find the pattern that addresses the particular problem. Buschmann et al. [2] defines a simple procedure for selecting a specific pattern based on their pattern categories, pattern description template and the relationships between patterns. Gamma et al provide a brief description about different approaches to finding a specific design pattern for a current design problem. Schmidt et al. [25] group architectural and design patterns into the following problem categories: Base-line Architecture, Communication, Initialization, Service Access and Configuration, Event Handling, Synchronization, and Concurrency. The categories partially overlap with the problem categories defined by Buschmann et al. [2]. The same pattern description template is used in [2] and [25], and it includes most of sections in [21] and also in a wide variety of pattern description schemata in common use today.

	WISA & Reference Architecture Deliverable ID: D4 B	Page : 30 of 53
		Version: 2.00 Date: 23 Oct 03
		Status : Proposal Confid. : Restricted

Architectural drivers can be determined by analyzing the purpose of the system and critical functional, quality and business requirements. They shape and drive the architecture of the system being built. Detailed investigation of particular aspects of the requirements may be needed in order to understand their implications for software architecture. At more detailed level, the drivers can be determined from the requirements on the particular subsystem or component. Architectural drivers enable the choice of architectural and design patterns.


Buschmann [13] uses a warehouse management system as an example. Multiple clients, many of which are mobile, are connected to the servers in the system inventory. The following architectural drivers have been identified from the non-functional requirements of this system: distribution, human-computer interaction, platform independence, and component integration. The application order of architectural patterns in this example is based on the importance of the architectural drivers.

The following steps for selecting a specific pattern have been applied from [2]:

1. Specify the problem and possible subproblems at hand, its or their forces, and associated architectural drivers. The term force is used to denote any aspect of the problem that should be considered when solving it, such as the requirements, constraints, and desirable properties of the solution.
2. Select the pattern catalog and specify the pattern and problem categories that will be used in the selection process.
3. Select the pattern category that corresponds to the current design phase and model.
4. Select the problem category that corresponds to the general nature of the design problem. If there is no appropriate problem category, then select an alternative problem category if possible (step 8)
5. Select the patterns and pattern variants whose problem descriptions and forces best match the design problem at hand and the combination its forces and architectural drivers. If no pattern matches the current design problem, then select an alternative problem category if possible (step 8) or investigate if a pattern, when specialized, can help to solve the problem.
6. Compare the consequences of applying the selected patterns. Investigate and compare what quality attributes the selected patterns help to achieve.
7. Select the pattern or patterns that provides the best solution to the current design problem. If several patterns were selected, begin with the pattern that addresses the most important aspect. If there are no encountered problems, the pattern selection ends.
8. Select an alternative problem category and return to step 5, if no problem category sufficiently matches the current design problem or if the selected problem category does not include suitable patterns. The alternative category may include patterns that can be specialized for the current design problem. If there is no new appropriate problem category alternatives, then select an alternative pattern catalog if possible and go to step 2. If there is no new alternative catalog, the design problem has to be solved without applying patterns because suitable patterns are not available.
9. Select an alternative pattern catalog and repeat the selection procedure using the pattern and problem categories of the catalogs.

The following sections of a pattern description particularly enable the application of the pattern selection procedure: Name, Intent, Problem, Context, Forces, Solution and Consequences. The sections can be identified, directly or indirectly, in most of pattern description schemata in common use today. The procedure can also help to select the patterns and styles that are not documented using any template of this kind.

The pattern descriptions in [2] and [25] document also the relationships between the patterns. The patterns connect, complement, and complete each other to form pattern languages. Pattern languages have become more popular than pattern catalogs because they have been found to be the most promising way to document sets of closely related patterns [25]. The pattern descriptions in [2] and [25] document also the relationships between the patterns. The patterns connect, complement, and complete each other to form pattern languages. This results a synergistic effect: the resulting pattern language is more than the sum of

	WISA & Reference Architecture Deliverable ID: D4 B	Page : 31 of 53
		Version: 2.00 Date: 23 Oct 03
		Status : Proposal Confid. : Restricted

its constituent patterns. The proper integration of pattern languages is also very important, but weakly addressed by publications.

6.1.3 Representing the proper application order of patterns

UML class diagrams have been used to describe associations between patterns. Schmidt et al. [25] propose a simple diagram for describing the application order of patterns. Rectangles in the diagram denote patterns. A pattern points another pattern with an arrow, if it uses the pattern in the implementation of its feature indicated by the name of the arrow. Shaded architectural patterns indicate the starting points of the pattern language.

The UML Notation Guide defines the Usage Dependency with the "use" keyword for a situation in which one model element requires the presence of another element for its correct implementation or functioning. Therefore, this dependency relationship has been used in Figure 6 and Figure 9 for describing the application order of patterns.

Brochers [26] proposes a formal notation for pattern languages. As the diagram of [25], the notation represents a pattern language as a directed acyclic graph whose nodes are patterns and edges references from one pattern to another.


6.2 APPLYING PATTERNS TO CONCEPTUAL ARCHITECTURE

Conceptual architecture is closest to the application domain because it is least constrained by the software and hardware platforms. The first step of creating an architectural vision is to identify high-level and system-wide architectural drivers that drive the conceptual architecture of the system being built. The drivers can be determined by analyzing the purpose of the system and critical functional, quality and business requirements. Detailed investigation of particular aspects of the requirements may be needed. The most important system-wide requirements characterize interaction styles and constraints between conceptual components in different deployment units. Two examples of high-level architectural drivers are:

1. "Terminals are user interface systems that exchangeable and location and migration transparent. They rely on powerful re-configurable computers for resources, such as files, devices, processing power and application and management software services."
2. "The application software component should be able to be flexibly deployed on terminals that can play the role of both clients and servers, depending on whichever role is needed for the task at hand."

The second step of creating an architectural vision is to select architectural patterns that address these properties. It is important to follow a specific design philosophy for achieving a property and select an architectural pattern or a set of patterns that support the implementation of the property. Structural aspects are generally more important than functional aspects when specifying a conceptual architecture for a system. Many architectural patterns are combinations of several elementary patterns and styles and thus may support combining several properties in the architecture being built. There may be available domain-specific patterns that provide typical system structures in a particular application domain and their underlying design principles.

The component and connector style is often used to model a collection of independently executing distributed components. Decomposable and interconnected components in the component and connector style provide a significant potential for reuse and for incorporating COTS components. It is important to isolate volatile communication and control aspects from reusable components. They can be isolated in connectors or separate control components can also be used in systems where control aspects are important from the conceptual architecture onwards.

	WISA & Reference Architecture Deliverable ID: D4 B	Page : 32 of 53
		Version: 2.00 Date: 23 Oct 03
		Status : Proposal Confid. : Restricted

Architecting usually begins with the partitioning a system into conceptual subsystems and high-level conceptual components using the decomposition style. The application of the pattern selection procedure (Section 6.1.2) with the pattern catalog in [21] and the above architectural driver 1 results in the selection of the N-tier Client-Server style [21]. 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 in a distributed wireless system. The application of the pattern selection procedure with the pattern catalog in [21] and the above architectural driver 2 results in the selection of the Peer-to-Peer style [21]. In the Peer-to-Peer networking architecture, each member node can make information available for distribution and can establish direct connections with other member nodes to download information. The two styles offer two general alternatives for the conceptual architecture of wireless systems

The Pipes and Filters architectural pattern [21] can be used for structuring distributed systems that process a stream of data if it is more often used for structuring the functional core of applications. The Reflective Blackboard architectural pattern [27] is the result of the combination of two other well-known architectural patterns: the Blackboard pattern [21] and the Reflection pattern [2]. The pattern allows a better separation of concerns, supporting the separate handling of control strategies by means of the computational reflection technique. The control strategies are handled independently from the application logic and data, providing a better architecture for large-scale multi-agent software. The pattern provides, early in the architectural design stage and also in conceptual architecture, the context in which more detailed decisions related to system-level properties can be made in late stages of software development.

If several pattern alternatives have to be applied for the conceptual architecture, begin with the architectural driver that addresses the most important architectural aspect and the pattern that supports the driver (One pattern at a time [13] in Figure 6). For example, both the N-tier Client-Server style and the Peer-to-Peer style may be applied to the same architecture. The deployment style in the allocation category is used to model potential allocations of conceptual structural elements to the conceptual execution nodes for the system context.


6.3 APPLICATION EXAMPLES OF PATTERNS TO CONCRETE ARCHITECTURE

The application of architectural and design patterns to various problems in designing concrete architecture for wireless systems will be discussed in this section.

6.3.1 Decomposing the software into modular parts

The selected and applied styles and patterns for the conceptual architecture specify fundamental overall structuring principles of concrete architecture, too. Architecting concrete architecture usually begins with the continuation of the system partition using the decomposition style. Many modern architecture design approaches partition the software into two highest level partitions or modules: application software into the applications module and infrastructure software into the infrastructure module (Figure 7). The rationale for this is that application software and infrastructure software will often change independently and they have distinct supplier value chains in the software industry [28]. This partition is similar to the partitions of the software into the vertical and horizontal partitions or into the domain dependent and domain independent partitions. The application, vertical and domain dependent partitions are similar. They include functionality that is central to the business. Accordingly, the infrastructure, horizontal and domain independent partitions correspond to each other.

The Three-tier Client-Server style [21] has been used in Figure 7 to decompose the software into the client and server tiers that communicate in the client-server fashion via the Middle tier. A software system may also be partitioned according to the most-important quality attributes, for example into a high-reliable part for high-reliable components, and into a high-performance part for high-performance components. The deployment style in the allocation category is used to allocate the elements of the Three-tier Client-Server style to the elements (nodes) of a hardware environment in Figure 7.

	WISA & Reference Architecture Deliverable ID: D4 B	Page : 33 of 53
		Version: 2.00 Date: 23 Oct 03
		Status : Proposal Confid. : Restricted

6.3.2 Decomposing the software into layers

The Layered style [21] helps to construct software architecture with the portability, exchangeability, reusability, and maintainability quality attributes. Examples of the architectural drivers that may result in the selection of the Layered style:

- Components should be replaced by alternative implementations without affecting the rest of the system.
- A low-level platform may be subject to change in the future.
- Other systems may later use same low-level issues.

Figure 7 shows that the same layered architecture of infrastructure software has been applied for each node. Generally, there may be a many-to-many relationship between modules, layers and run-time structures and components. This means for example that an infrastructure module may include one part that corresponds to a device driver in the Device Drivers layer and another part that correspond to an operating system in the Operating Systems layer.

6.3.2.1 Decomposing middleware using the Layers pattern

Schmidt [31] uses the Layers pattern [2],[3] to decompose middleware into multiple layers and describes R&D efforts related to each layer. Those layers can also be considered as a structure for commercial middleware components. The layers are the following: host infrastructure middleware, distribution middleware, common middleware services, and domain-specific middleware services (Figure 7).

The host infrastructure middleware layer encapsulates portable and reusable components that abstract away the incompatibilities of individual operating systems. Next-generation middleware should be adaptable to changes in underlying operating systems and networks and customizable to fit into devices from PDAs (Personal Digital Assistant) and sensors to powerful desktops and multicomputers [29]. Some host infrastructure middleware components are available for real-time and embedded systems.

Distribution middleware augments the host infrastructure by defining higher-level distribution programming models. QoS-enabled object request brokers (ORBs) are at the core of distribution middleware. Some ORBs are available for real-time and embedded systems.

Common middleware services extend distribution middleware by defining higher-level domain-independent components that focus on allocating, scheduling, and coordinating various end-to-end resources throughout distributed systems. Traditional middleware glues together distributed application components and its support for evolving software requirements relies only on component interfaces. The role of middleware in many mobile systems is less to glue components than to enable dynamic interaction patterns between autonomous concurrent components [30]. Some open source services at common middleware services layer are available for real-time and embedded systems.

Domain-specific middleware services target vertical markets, unlike the previous three middleware layers, which provide horizontal services. Domain-specific middleware services are the least mature of the middleware layers from the point of view of COTS components. However, they have great potential for cost-effective COTS components.

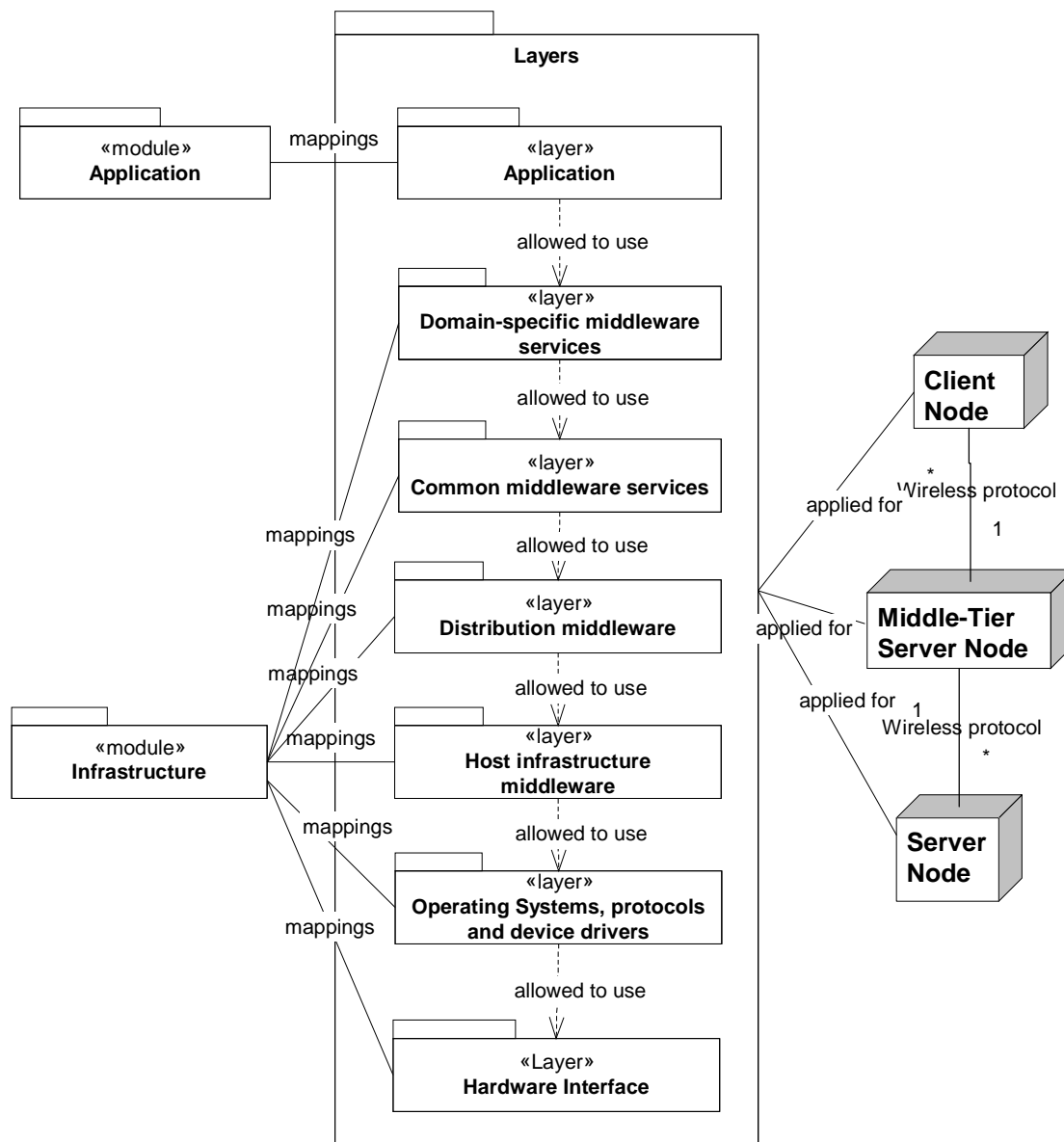



Figure 7. The software system is partitioned into two high level modules and seven layers, and allocated to three distributed nodes.

6.4 APPLYING PATTERNS TO DISTRIBUTED SYSTEMS AND MIDDLEWARE

6.4.1 Middleware

Middleware facilitates the collaboration of components and services in heterogeneous distributed environments. At the heart of middleware are Object Request Brokers (ORBs), such as CORBA, DCOM, and Java RMI, which automate many tedious and error-prone distributed programming tasks. Middleware standards have matured considerably with respect to the requirements of distributed real-time and embedded systems [31]. For example, the Minimum CORBA renders CORBA usable in memory-constrained

	WISA & Reference Architecture Deliverable ID: D4 B	Page : 35 of 53
		Version: 2.00 Date: 23 Oct 03
		Status : Proposal Confid. : Restricted

applications. The Real-Time CORBA supports applications that reserve and manage the network, CPU, and memory resources predictably. The CORBA Messaging provides timeouts, request priorities, and queuing disciplines to applications. The Fault-Tolerant CORBA supports replication, fault detection, and failure recovery. Robust implementations of these CORBA standards are available. In addition, there are emerging standards such as the Dynamic Scheduling Real-Time CORBA, the Real-Time Specification for Java, and the Distributed Real-Time Specification for Java.

Commercial middleware technology provides superior programming environments for wireless Internet solutions. However, its current specifications cover a narrow subset of useful Internet protocols only [32]. Software service architectures need to be revised. Context-awareness challenges location transparency. The client-server, request-reply and remote procedure call (RPC) models are not enough in the wireless world. The ROBOCOP project [33] has defined a component based SW architecture for the middleware layer of high volume embedded devices such as cell phones, PDAs as well as internet and broadcast terminals like set top boxes, network gateways, digital television sets. Commercial middleware technology is not yet mature enough to cover large-scale dynamically changing systems. Pree and Pasetti assess existing middleware such as CORBA and its real-time extension as being too complex for real-time control systems and propose a lean middleware concept instead of it [34].

Conventional ORBs are hard to port, optimize and evolve because they are not dynamically configurable. Schmidt and Cleeland [35] illustrate how a pattern language can be used to develop extensible ORBs and quantify the positive impacts of applying this pattern language. The standard CORBA interoperability protocols are not suitable for applications with stringent message footprint size, latency, and jitter requirements. Schmidt et al. [36] describe how to apply patterns to develop a pluggable protocols framework for ORB middleware. Pluggable protocols frameworks are needed to replace standard CORBA interoperability protocols in applications with stringent message footprint size, latency, and jitter requirements.

6.4.2 Basic building blocks when working with RPC middleware

Three fundamentally different paradigms are used to bring apart software objects from each other in distributed systems: the remote procedure call (RPC) paradigm, the posting and receiving messages paradigm and the continuous streams of data paradigm [37]. The client-server style is often used in RPC systems.

RPC (remote procedure call) middleware allows clients to communicate with objects on a remote server. Völter et al. [37] discuss basic infrastructure building block patterns and components for working with (or constructing) an object-oriented RPC middleware. An example set of basic building blocks when working with RPC middleware is shown in Figure 8 [37].

The Client Proxy object within the client process offers the Remote Object's interface for a client. An Interface Description object defines the interface. The client uses a Client Request Handler to handle network communication.

On the server side, a Server Request Handler receives the remote calls invoked by the client. It forwards invocations to the Invoker object, after the message is received completely. The Invoker dispatches the message to the responsible Remote Object using the received invocation information.

Marshaller objects are used to serialize and de-serialize complex types on the client and server nodes. The Lifecycle Manager manages lifecycles of Remote Objects.

A server application creates and controls the Remote Objects. An application is comprised of a set of related Remote Objects. Several applications may run in one server process. A Framework Facade component provides the designer with the single access point and administration API to the RPC

middleware and its services. The client and server Request Handlers use a Communication Framework component in a lower level layer to handle low-level details of network communication. The developer can use a Protocol Plug-in component to exchange or adapt the protocol implemented by the client and server Request Handlers.

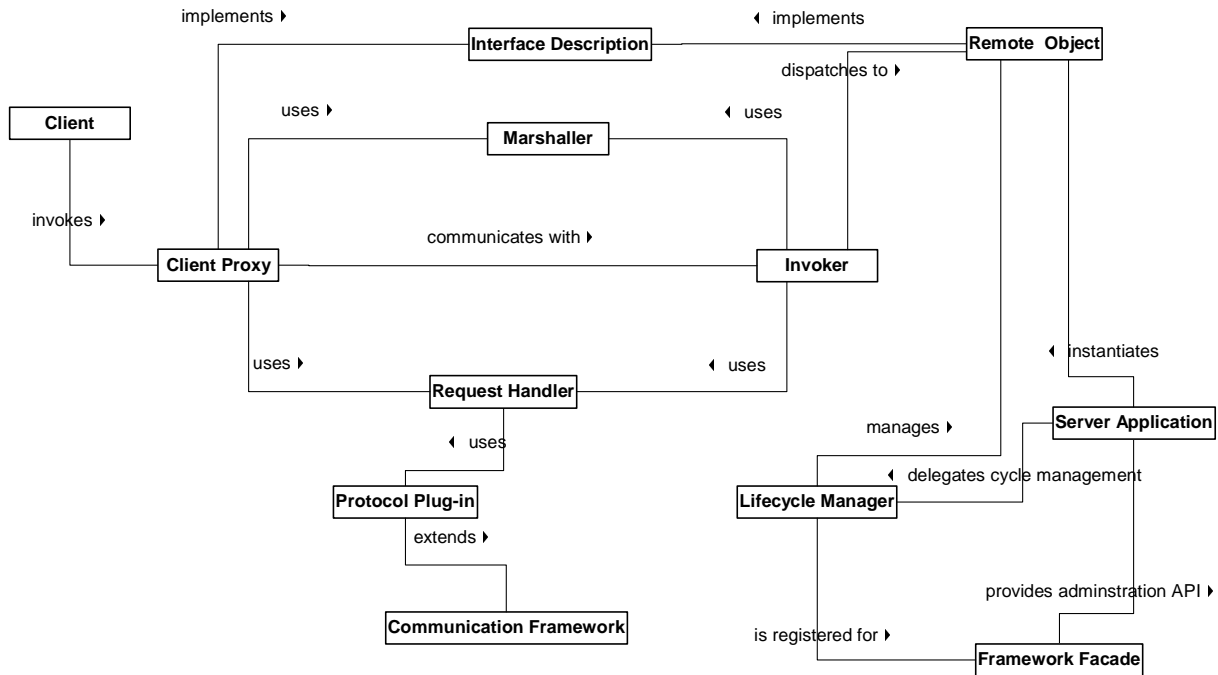



Figure 8. Basic building blocks when working with RPC middleware (redesigned from [37])

6.4.3 A pattern language for building networked systems and middleware

The application order of patterns for constructing concurrent and networked applications is shown in Figure 9. A pattern points another pattern with a dependency arrow with the "use" stereotype, if it uses the pattern in the implementation of its feature indicated by the name of the arrow. Shaded architectural patterns indicate the starting points of the pattern language.

The Broker architectural pattern [21] can be applied to integrate the conceptual architecture with the components that responsible for co-coordinating distributed communication. Client Proxies, Request Handlers and Invokers in Figure 8 build together a broker. The Proxy pattern [2] is used to make clients of a component communicate with a representative proxy rather than to the component itself. The Model-View-Controller pattern [21] can then be integrated to this structure if a large variety of user interfaces is needed.

The Half-Sync/Half-Async architectural pattern [25] can be used to introduce two intercommunicating layers to the broker component of the Broker pattern, one for asynchronous and one for synchronous service processing and a queuing layer for mediating the communication between services in the other two layers. The Proactor pattern [25] can help implement the asynchronous service processing layer in event-driven applications, particularly servers, that receive and process requests from multiple clients concurrently. The Active Object pattern [25] and Monitor Object pattern [25] can be used to implement the queuing layer. The Half-Sync/Half-Reactive variant of the Half-Sync/Half-Async architectural pattern can be implemented by

	WISA & Reference Architecture Deliverable ID: D4 B	Page : 37 of 53
		Version: 2.00 Date: 23 Oct 03
		Status : Proposal Confid. : Restricted

combining the Reactor pattern [25] with the Thread Pool variant of the Active Object pattern. The Reactor pattern helps structure event-driven applications, particularly servers that receive requests from multiple clients concurrently but process them iteratively. The Acceptor-Connector pattern [25] decouples connection establishment and service initialization from service processing. It helps ORB connection management. In Figure 9, the pattern specifies that there are three types of event handlers: acceptors, connectors, and service handlers.

The Interceptor pattern [25] can be used to implement a support for out-of-band extensions in distributed software architectures. The Component Configurator pattern [25] can be used to configure concrete interceptors or event handlers into a networked system dynamically at run-time.

The Leader/Followers architectural pattern [25] can help implement high-performance multi-threaded servers in the multi-tier client-server architecture. At the heart of this pattern is a thread pool mechanism that can be implemented using the Monitor Object pattern. The Reactor and Proactor patterns can be used to demultiplex and dispatch events from event sources to event handlers.

The Wrapper Facade pattern [25] is used to abstract low-level system calls and to encapsulate the function and data provided by existing non-object-oriented API's. Wrapper Facade components are in the host infrastructure middleware layer in Figure 7.

The Reactor pattern can be used instead of the Leader/Followers pattern when each event requires a short amount of time to process. The Proactor pattern can be used in lieu of the Leader/Followers pattern when an operating system supports asynchronous I/O effectively and when designers master the asynchronous inversion of control associated with the Proactor pattern. The Half-Sync/Half-Async and the Active Object patterns may be used instead of the Leader/Followers pattern when requests in a queue must be reordered before they can be processed by threads in the pool and or when event sources cannot be waited for by a single event demultiplexer efficiently. [25]

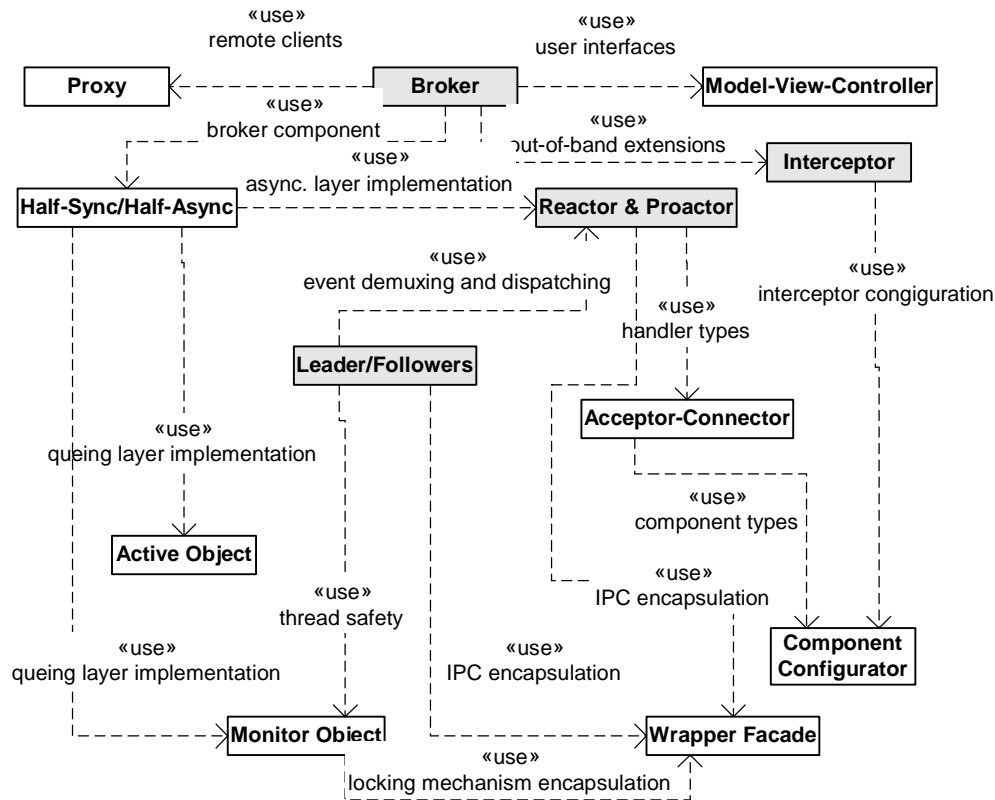


Figure 9. The application order of patterns for networking concrete architecture (re-designed from [25])


6.4.4 Examples and experiences

- **Patterns for distributed real-time systems and embedded middleware**

Patterns for designing and optimizing distributed real-time systems and embedded middleware have been published in the following workshops and conferences: The OOPSLA 2001 workshop [56], PLOP 2002 [57], and the OOPSLA 2002 workshop [58]. They are a good addition to the patterns in [25], and some of them will be briefly commented in this section.

Subramonian and Gill [38] present a preliminary clustering of patterns based on categories of design forces of patterns that are relevant to platforms of networked embedded software. The focus of the paper is unfortunately on wired networks. Experience from patterns in wireless networks has not yet been published in that research program.

Welch et al. [39] propose four architectural patterns for resource management in distributed systems. The Resource Instrumentation pattern helps to collect and maintain information about the recent state of computing and network resources. The Software Performance Monitoring pattern helps to determine and assess the status (e.g., performance) and resource needs of distributed real-time software systems. This pattern assumes that the real-time systems are able to send time stamps to a system management component. The Allocation Planning and management pattern helps to determine how

	WISA & Reference Architecture Deliverable ID: D4 B	Page : 39 of 53
		Version: 2.00 Date: 23 Oct 03
		Status : Proposal Confid. : Restricted

and when to allocate hardware resources to distributed software systems. The Resource Control pattern helps to perform the allocation actions of hardware resources to software systems.

Gill et al. [40] describe refinements to a pattern language for resource scheduling in distributed real-time systems. They also present application guidance of the language.

The Virtual Component design pattern [41] helps to reduce the memory footprint of middleware. This compound pattern applies the Factory Method [24], Component Configurator [25] and Proxy [2] design patterns. It uses the Factory Method pattern to defer the instantiation of an object to subclasses, and the Component Configurator pattern to provide a dynamic component configuration. The proxy component of the Proxy pattern provides a placeholder for another object to control access to that object. The lazy dynamic variant of the Virtual Component pattern applies the Proxy pattern to implement a component proxy and the Lazy Acquisition pattern to defer resource acquisition.

The QoS Contract pattern [42] helps to decouple QoS measurement, adaptation and management from a functional application. The Snapshot pattern [42] can be used to provide a useful approximation of the current state of a distributed system.


The Quality Connector pattern language [43] helps to define and provide quality-constrained services in distributed real-time and embedded systems. The Quality Connector pattern language and the Component Configurator [25] and the Virtual Component [41] patterns provide means to change the behavior of a service during run time.

Marinucci et al. [44] propose a set of patterns for designing reusable middleware components for QoS and resource management, and another set for engineering systems that employ middleware services. These two sets work together and provide a pattern language for engineering dynamic real-time applications.

- **Patterns in designing high volume configurable embedded appliances.** The ROBOCOP project [33] has defined a component based SW architecture for the middleware layer of high volume configurable embedded devices such as cell phones, PDAs as well as internet and broadcast terminals like set top boxes, network gateways, digital television sets. The Client Server [21] and Layers [21] patterns have had an important role in the development of the ROBOCOP software architecture.
- **A pattern for managing distributed workflows.** The Manage Distributed Workflows pattern [45] allows distributed workflows to be executed and managed in several workflow servers.
- **Maintaining consistent state between clients**
The Observer design pattern [24], also known as "Publish-Subscribe" [2] and "Dependents", can be used to solve the problem how to maintain consistent state between clients in a wireless system [49]. The Observer pattern provides a loose coupling between the Subject and Observer components by specifying interfaces that allow Observers to register for event notification and enable the Subject to notify its Observers of state changes.

The Abstract Session structural pattern [50] allows a server object with many client objects to maintain per-client state and maintain type-safety.

- **Adapter design pattern for handling interface mismatches**
The Adapter design pattern [24] can be used to solve the following problem: A client requires the use of a service provided by a server class, but the interface of the server does not match what the client expects. The adapter design pattern outlines the following solution: The client should interact with the server via an adapter class or component. The adapter adapts the interface of the client to a domain-specific interface that the client uses.

	WISA & Reference Architecture Deliverable ID: D4 B	Page : 40 of 53
		Version: 2.00 Date: 23 Oct 03
		Status : Proposal Confid. : Restricted

- **Architectural patterns for enterprise application integration**

Lutz [51] presents five architectural patterns for enterprise application integration. The Integration Adapter pattern helps to convert an existing application interface to a desired interface. The Integration Messenger pattern can be used to minimize application communication dependencies between applications. The Integration Mediator pattern helps to minimize application dependencies by encapsulating application integration logic. The Process Automator pattern helps to minimize dependencies between process automation logic and applications.

- **Patterns for the J2EE and Jini technologies**

15 core J2EE patterns [46] help to solve J2EE-specific architectural problems. The J2EE Patterns catalog addresses three tiers of the five tier J2EE model. The presentation tier patterns are related to Servlets and JavaServer Pages technology. The business tier patterns are related to the Enterprise JavaBeans technology. The integration tier patterns are related to the Java Message Service API and the Java DataBase Connectivity API. The J2EE patterns framework shows the interrelations between the fine-grained elements in each pattern. The framework helps the developer to understand the relationships among patterns as well as the implications of choosing a combination of patterns by documenting associations and dependencies between patterns. The framework also helps developer in choosing the proper combination of patterns when designing an end-to-end (presentation-business-integration) solution using the J2EE patterns.

Sun Microsystems classifies Java technology into three editions: Java 2 enterprise edition (J2EE), Java 2 standard edition (J2SE), and Java 2 micro edition (J2ME) [47]. J2EE focuses on enterprise server side development. J2SE focuses on desktop computing development. J2ME supports the development of consumer electronics and embedded systems. J2SE implements the Java 2 specification, while J2EE and J2ME are based on J2SE.

Submissions to the OOPSLA 2000 workshop 'The Jini Pattern Language' address patterns that can be used to solve design issues embodied in the Jini Technology. For example, the Profile-based Service Browsing pattern [48] can be used to design an architecture that supports discovering network services depending on user preferences and service and terminal capabilities.

- **Applying patterns to protocols**

- **Patterns for protocol system architecture.** The Protocol System pattern, the Protocol Entity pattern and the Protocol Behavior pattern [52] provide common principles for understanding existing protocols and their parts, or implementing new ones.
- **The use of the Strategy design pattern for protocols.** The Strategy design pattern [24] can help to compose reliable distributed protocols [53].
- **Applying the Adapter design pattern to adapting protocol stack interfaces.** The Adapter design pattern [24] can be used to design modular plug-in components that adapt the interface of each supported protocol stack interface to a generic interface that the application can rely on [54].
- **The use of patterns in the Bluetooth communication.** Gokhale [55] discusses the use of the following patterns in Bluetooth: Broker [21], Layered [21], Lookup, Bridge [24], Facade [24], Service Browser, Command [24], Interpreter [24], Dispatcher [25], Master-Slave [2], and On-demand Activation.

7. WISA REFERENCE ARCHITECTURE (WISA/RA)

WISA/RA provides a conceptual architecture that is used as a corner stone in the architecture development of a new service. This in turn assists service developers to find the services already known and concentrate the development effort on new services and components needed in the new end-user service. This section describes three views of the conceptual architecture of WISA/RA. A single concrete architecture for WISA/RA is not given but several typical concrete architectures are given in the WISA handbook (ref 3). The possibility of developing concrete views based on them will be considered in the next iteration.

7.1 CONCEPTUAL STRUCTURAL VIEW

The selected high-level WISA conceptual structure depicts the main domains of the WISA taxonomy as a layered architecture that also supports a direct interaction between nonadjacent layers if it is required by end-user applications (Figure 10). The service management services domain is shown on the side because its services interact with several components of the other domains to monitor and control the use of resources by a wireless service and its enabling services.

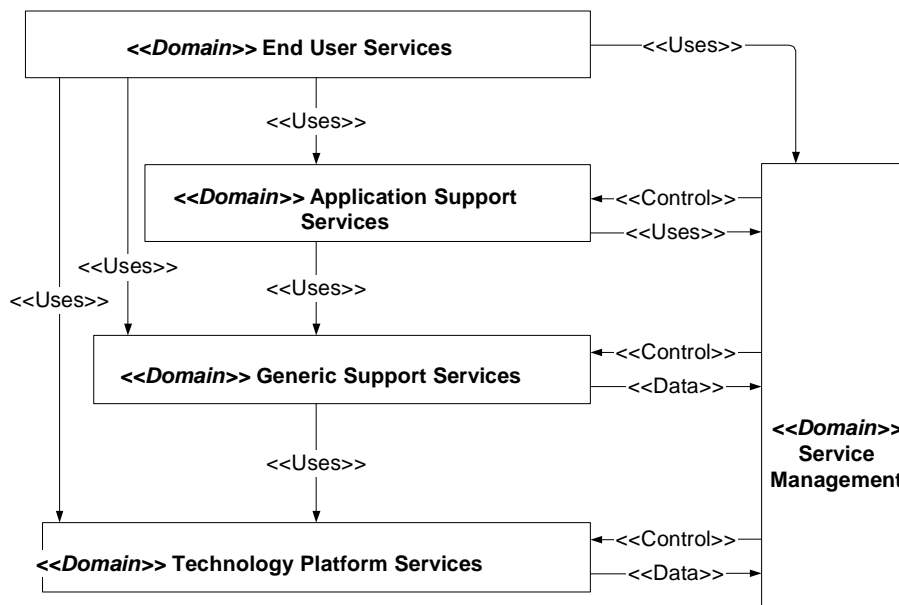


Figure 10. High-level conceptual structure of WISA/RA.

This layered structure can be used as the basis of the conceptual structure of any wireless service and its enabling services identified to belong to one or more of the domains. The services and components developed in the case studies of the WISE project, earlier projects and commercial components that are applicable building blocks for wireless services found out from different marketplaces were mapped to the layers of this conceptual structure. From the overall picture based on WISA taxonomy the scope of the WISA reference architecture was narrowed to three domains of most importance in wireless services (Figure 11). Therefore, the conceptual structural view of WISA/RA includes the basic services of which service engineers may have the greatest benefit; Application Support Services, Generic Platform Services and Service Management Services.

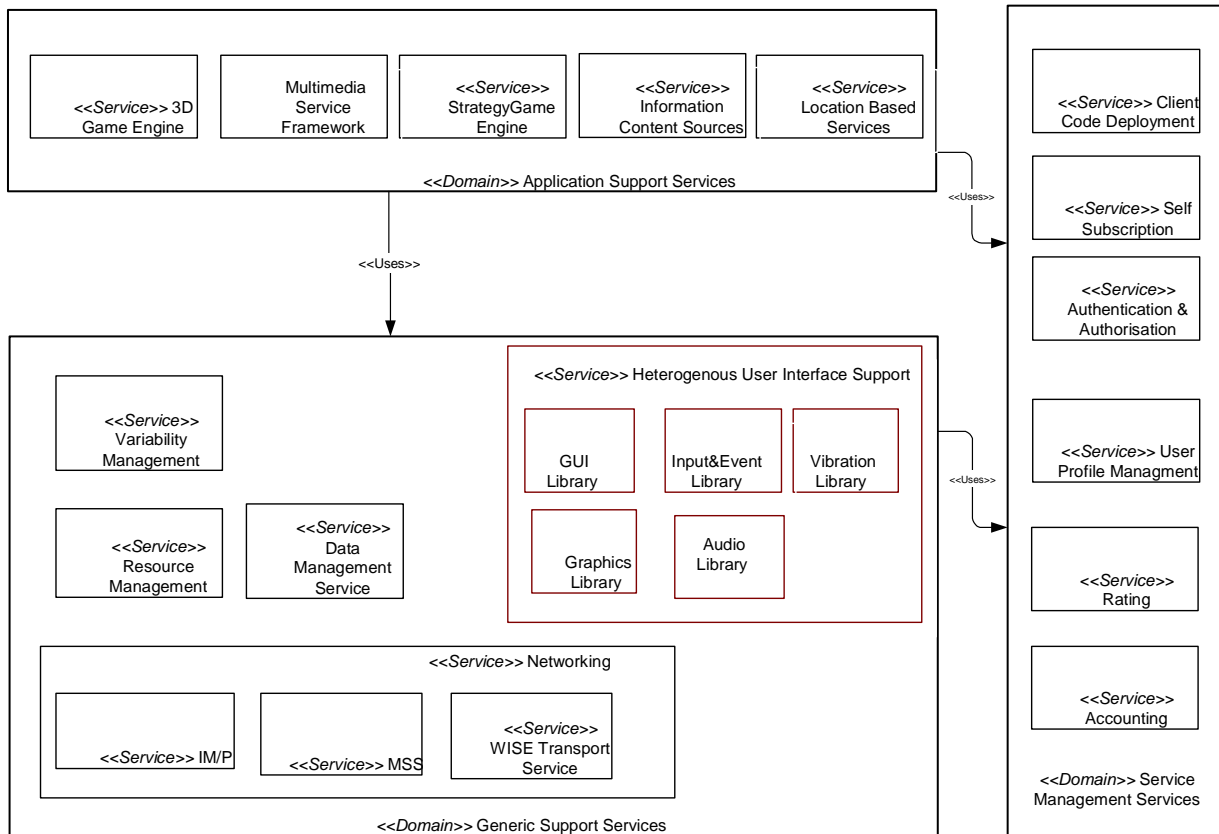


Figure 11. WISA conceptual structural view with some basic services.


Application developers use Application Support Services that are application-specific solutions, reusable inside the particular sub-domain. These services are mostly provided as application frameworks that boost modifiability and reusability. Generic Platform Services provide services shared by all wireless services including management level services, communication services and common components for user interfaces, mainly used by service providers, service developers and content providers. Service Management Services support service providers in provisioning and managing end-user services. The relations between support services inside a domain may vary based on the end user service or choice of specific support service implementations. These relations are shown for services in the handbook part of D4 (ref 3) for services in service catalog.

Application Support Services utilize the services of the other domains. Generic Platform Services are also dependent on Service Management Services and naturally on Technology Platform Services that are out of the scope of the WISA/RA but the relations between these domains are visible in the detailed descriptions of WISA basic services.

Potential reusability is highest in the domains of Generic Platform Services and Service Management Services. Therefore, there are also possibilities to use third party components if their functional and quality properties meet the requirements set by the end-user service, WISA/RA and further more, the domain the component belongs to. There is no unambiguous reasoning, which styles and patterns best suit to the particular sub-domains, e.g. resource management services or communication services.

The layered style was selected to the main style of WISA/RA due to

- 1) The quality attributes it supports (maintainability, modifiability, reusability and portability),

	WISA & Reference Architecture Deliverable ID: D4 B	Page : 43 of 53
		Version: 2.00 Date: 23 Oct 03
		Status : Proposal Confid. : Restricted

- 2) Simplicity,
- 3) Familiarity of the style among the service engineers, and
- 4) Popularity of the style in existing assets (i.e. services components and tools).

In order to achieve the most important quality attributes set to WISA/RA (cf. ref. 2) the main architectural style should support as much quality attributes as possible. Simplicity makes it ease to understand, and familiarity guarantees its suitability as a negotiation means with the managerial staff of customers and all business stakeholders. The last argument may be the strongest one for the use of a layered style. The main goal of WISA/RA is to organize the existing artifacts and assets to a form that assist their community-wide use, not to promote a new style that set obstacles to using the existing Internet based solutions in the wireless world. Strict layering is not enforced to ensure flexibility of reference architecture in emerging service categories when domain specific functionality needed in application support services or even generic support services is not yet stabilized and understood.

The layered style is often trade-off against performance of the system. The performance, however, is not considered in the conceptual view of the WISA reference architecture because the degree of layering at concrete level of architecture may be different i.e. several conceptual layers may be combined into single layer when designing the concrete architecture.

7.2 CONCEPTUAL DEPLOYMENT VIEW

The deployment view of WISA overall conceptual structure (Figure 3) derived from WISA taxonomy is presented in Figure 12. For simplicity the links inside a deployment unit already shown in high level conceptual structure view are left out. The link from service management services to business processes is shown to visualize their relation to WISA but the details are left out because they are not usually relevant for service developers. Note that the service management services do not need control the supporting services used in service management server unit, only those used by the end-user service directly or indirectly.

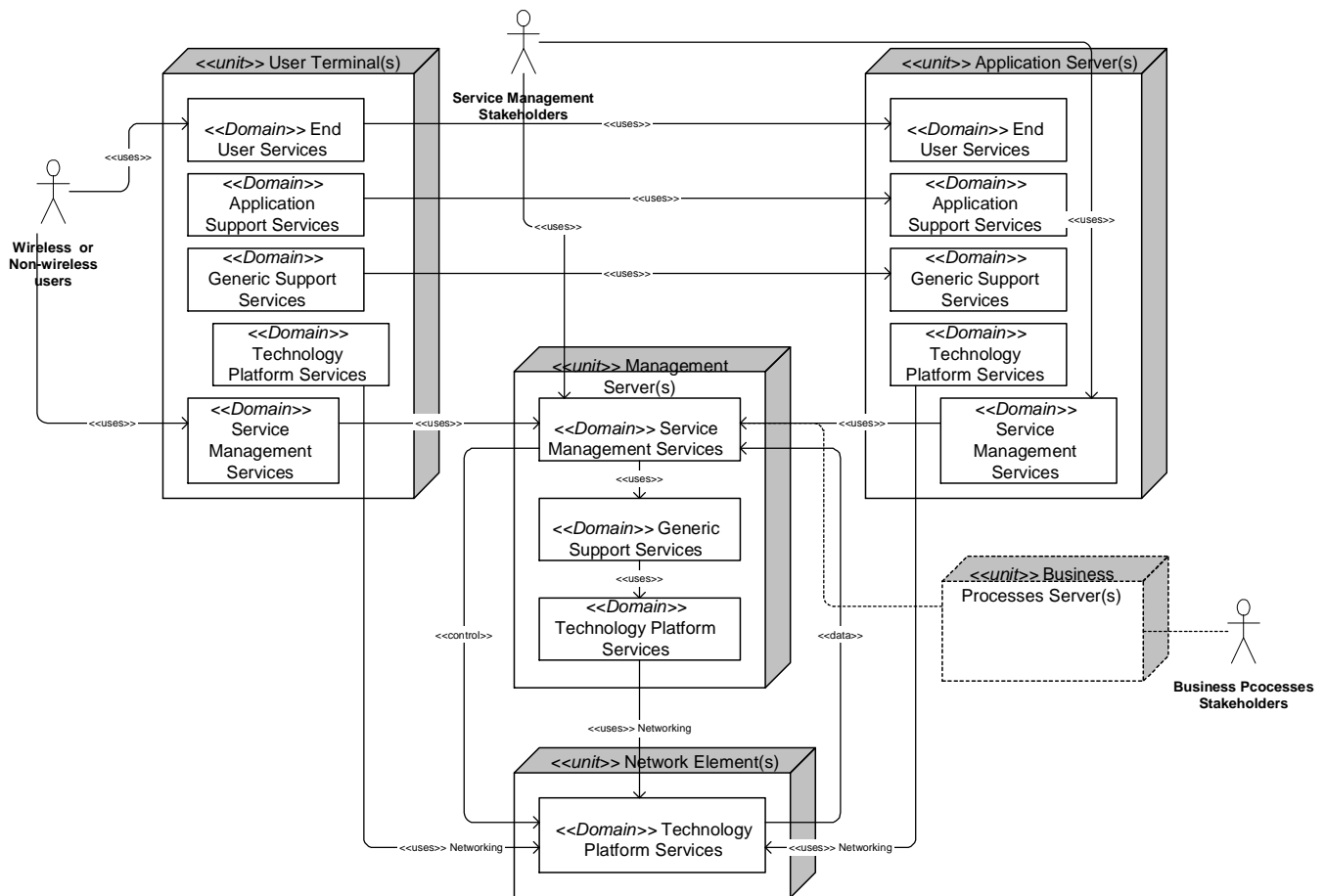


Figure 12: Conceptual deployment units of WISA

The types of deployment units are described in Table 11.

Table 11. Deployment units of WISA.

Deployment Unit	Description
Terminal	Wireless or non-wireless terminal used by the end-user or a terminal containing hardware control/sensor functionality. A terminal is wireless if it only has wireless communication channels.
Application Server	Unit containing service functionality coordinating service in several terminals.
Management Server	Unit containing service management functionality common to several services.
Network Element	Unit containing wireless or fixed networking functionality between platforms in several deployment units (for example GSM network elements).
Business Processes	Unit containing functionality to link wireless services to business processes (billing etc.)

This generic deployment shows typical deployment units and can be customized by combining two or more of units into single unit (which requires the integrability of support services required by them). Note also that this deployment is independent of architectural styles selected for the end user service, service management services or in fact any of the services in the WISA taxonomy. For example the end user service can be deployed using client-server or p2p style. The middleware or technology platforms in the separate nodes, also of the same type, can differ. Also, for example, the service management services are

often not used by the end-user directly and can be left out from the end-user terminal (or only monitoring/control function of terminal resources deployed).

The deployment of a typical end-user service is shown in Figure 13. The service management functionality in terminal is shown as separate service. The service client and management client can be integrated into one component in the concrete architecture. Examples of separate management clients from service client are SMS message based configuring of new features (easy to handle billing) for the service or separate management interface in a fixed (web browser based) terminal. Generic support services are often not available as identifiable services but as platform specific implementations. The networking and business processes deployment units are usually transparent for service developer (hidden by technology platform services or management services. The choice of terminal types narrows down the possible platform choices. The characteristics of wireless client platforms and software design issues are studied in more detail in deliverable D3 of Wise project, Management of heterogeneous clients (ref 4).

The internal deployment of service depends on the on the selected architectural pattern. The client-server division in Figure 13 should not be confused with client-server style. Here the service client in terminal can contain also server functionality, for example in a special case the terminal contains only a hardware or environment monitoring server. Similarly the communication and deployment of service between terminals or application servers depends on the server side architectural style. Usually one of the application servers is a gateway for load balancing. A third party application support web service (for example a content service) can be used by the application.

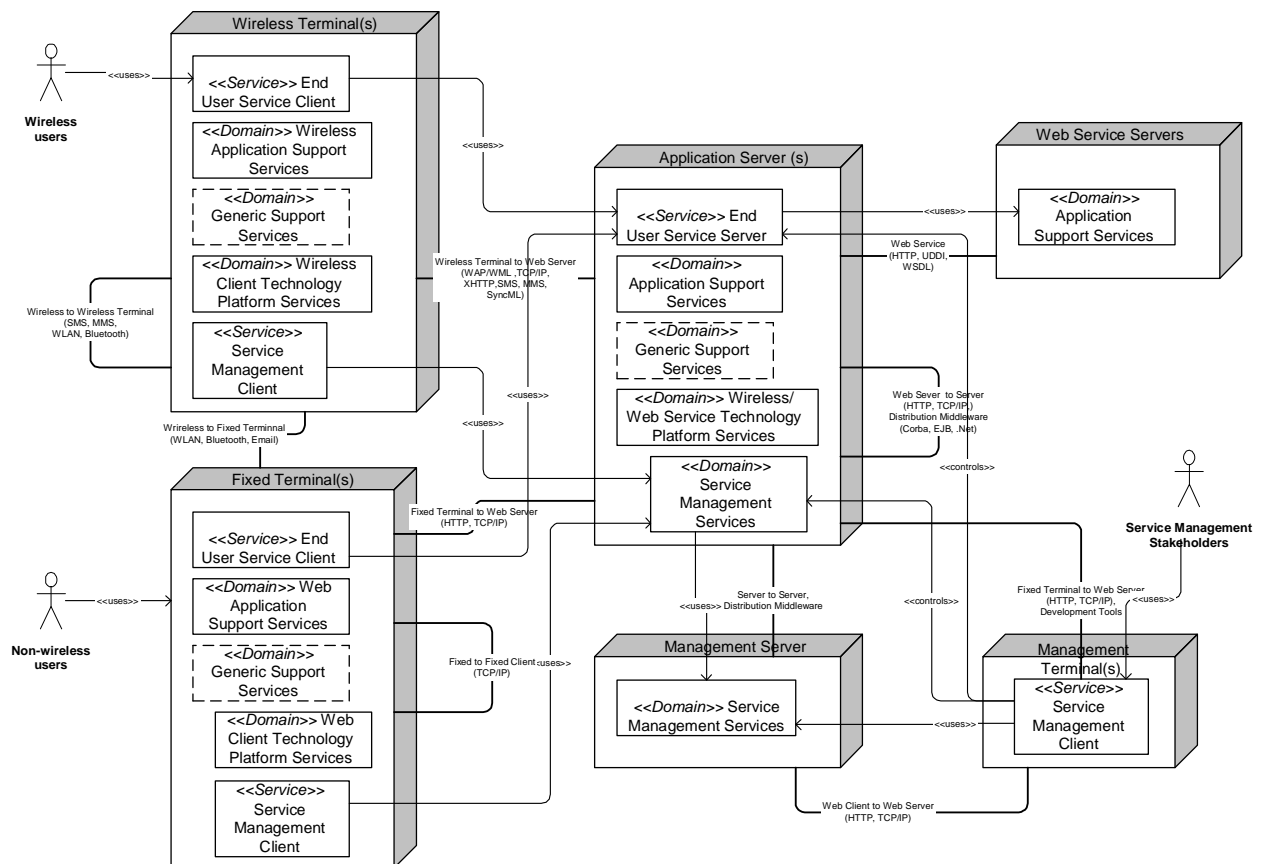


Figure 13. Deployment of an end-user service.

The communication links between different types of nodes are shown in Table 12.

Table 12. Types of communication links in WISA..

Communication Link	Description
Wireless to wireless terminal	Direct communication between wireless terminals (rich call, voice, image transfer, text messaging etc., the communication type is restricted by the communication technology).
Wireless to fixed terminal	P2p communication between wireless and fixed terminals (content downloading etc).
Fixed to fixed terminal	P2p communication between fixed clients (content sharing or messaging between users etc.).
Wireless terminal to Web server	Wireless use of a service (Multiplayer game playing, news reading etc.)
Fixed terminal to Web server	Web browsing (service control, content downloading etc.)
Server to server	Web based communication between distributed servers (integration of different server side platforms etc).
Distribution middleware	Middleware based communication between distributed servers (integration servers using compliant middleware technology).
Web service	Web service (Use of third party information content etc.)


The quality concerns in choosing the deployment view were mainly flexibility and integrability. Selecting the same layered style inside the each of the deployment units helps to ensure both of these. Again this is a not a tradeoff against performance at conceptual architecture level. At conceptual level the performance depends on the deployment of components inside each individual service (i.e. end-user or support service) rather than the layering. Because of this these deployments are not shown in the reference architecture but will be considered in the deployment view of each specific service in the service catalog or typical architecture in the WISA handbook (ref 3).






7.3 CONCEPTUAL BEHAVIOUR VIEW

To be considered in iteration 3.

7.4 CONCEPTUAL DEVELOPMENT VIEW

The development view of WISA/RA presents the source of the services of the sub-domains and their readiness (Figure 14). The following color-coding illustrates the status of the sub-domain:

	WISA & Reference Architecture Deliverable ID: D4 B	Page : 47 of 53
		Version: 2.00 Date: 23 Oct 03
		Status : Proposal Confid. : Restricted

- 
 Services (and components) defined to this sub-domain are ready for use
- 
 There are some services already ready but some are under development or will be developed later
- 
 Services are out of the scope of WISA/RA, e.g. hardware environments in the domain of Technology Platform Services
- 
 Services will be defined and implemented later but they are included to the scope of WISA/RA.
- 
 Services of the sub-domain are under development

The aim of the development view is to support the development of wireless services in work allocation, managing and controlling, and record relations that require collaboration between the service developers and users. Due to better management support the main domains are divided into sub-domains that present the hierarchical structure of the WISA/RA as topology diagrams (ref. 1) (Figure 15, Figure 16). On the contrary to the structural view that presents services as the building blocks of main domains, the development view presents aggregation levels more thoroughly to manage the features of a set of services. The reason is the generalization of services that can be made by clustering similar features to the same domain and defining the variation points for adding variable features to the common ones. Thus, the development view is necessary for the evolution of WISA/RA. The results of generalization are visible in the structural view of WISA/RA and in the descriptions of its basic services.

Managerial staff of the service provider and the wireless service developers is the main users of the development view. However, the domain and product line architects are responsible its continuous development. It is also the way to connect WISA/RA to Technology Platform Services, and check that WISA/RA conforms to the properties of existing wireless platform technologies and the standards used in that domain.

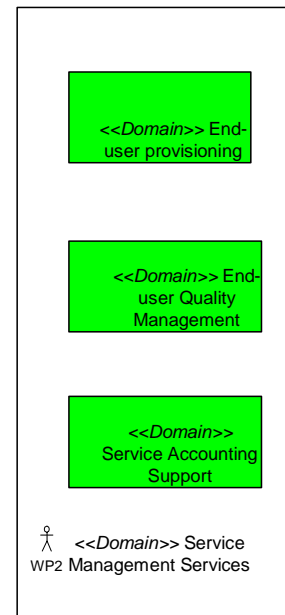
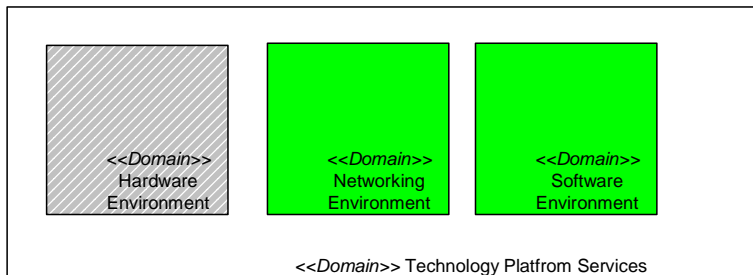
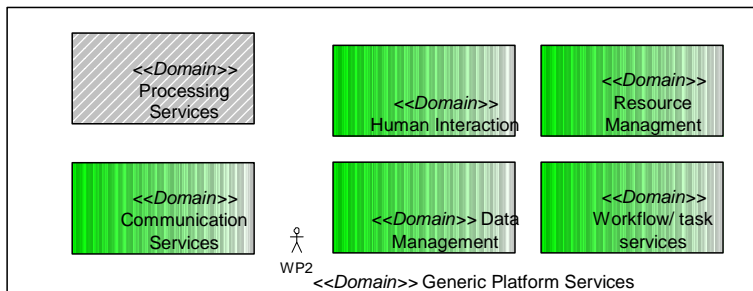
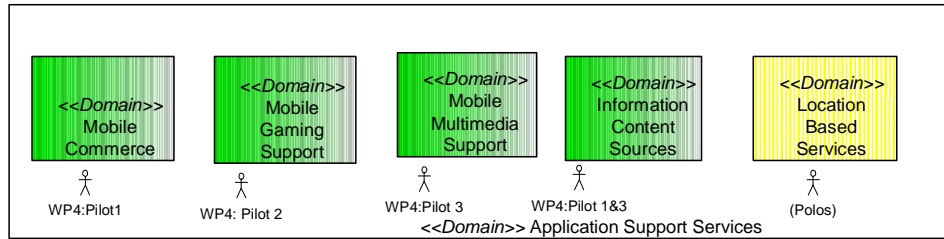


Figure 14. Topology diagram of WISA service domains.

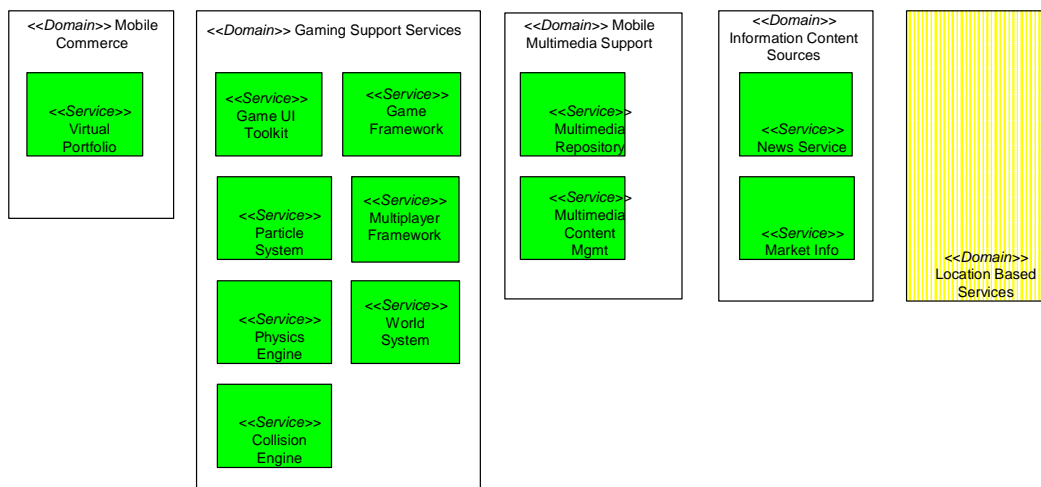


Figure 15. Topology diagram of Application Support Services.

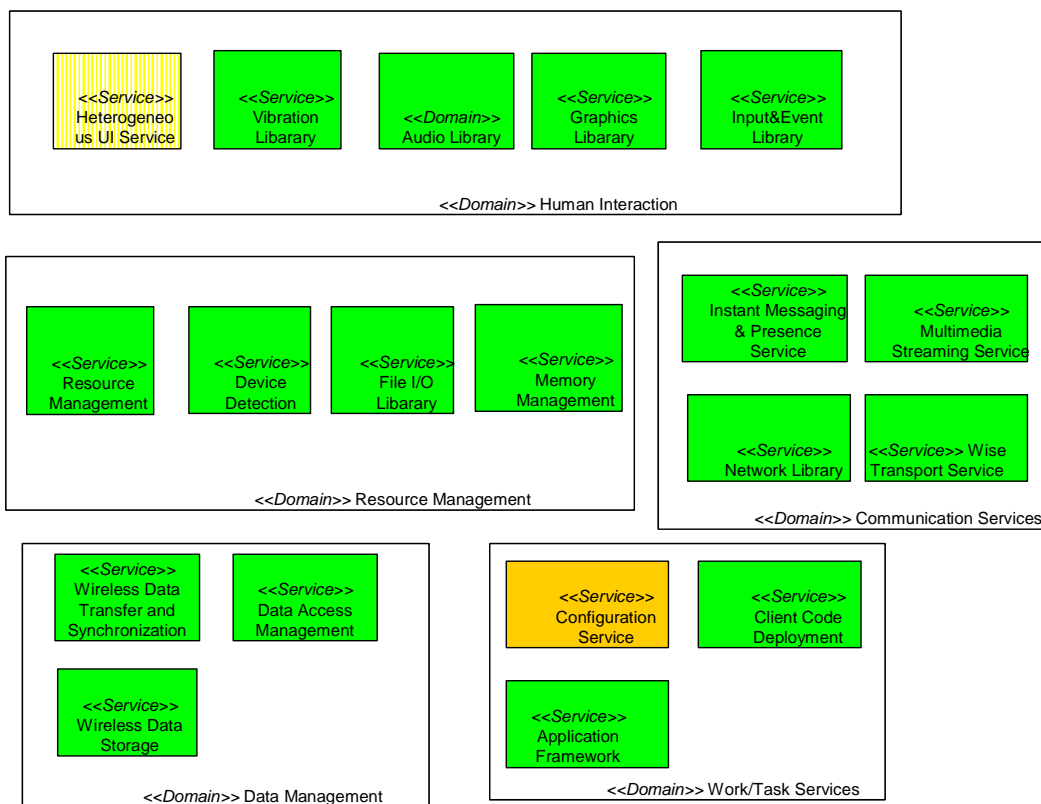



Figure 16. Topology diagram of Generic Platform Services.

A service in the service catalog belongs to one of the taxonomy domains. End-user services will not be included into WISA service catalog. For technology platform services only their selection criteria will be presented. Each service in the catalog is presented according to the following documentation pattern:

1. Identification of a <<service>> describes the name and type of a service. The type links the service to a taxonomy domain.
2. An overview describes the purpose of the service.
3. Source of the service.
4. Special terms and rules define what styles, patterns and constraints a service has to follow.
5. Quality attributes and how they are met.
6. Provided features describe the features provided by a service and variability of these features.
7. Required features describe the features required from the other services of the systems in the use of a service.
8. Conceptual structure of a service.
9. Conceptual deployment of a service, as an example.
10. External component diagram of the service (to be included into library),
11. Interfaces provided by the service, abstract message descriptions.
12. Required interfaces, named & from which component.

The listed items (1-7) are required to map the service to the structural view of WISA/RA. The items 8-9 are required to design and analysis of the conceptual architecture of a wireless service. The rest of information is used in the development of the concrete architecture of a wireless service (this concrete part of specification will be developed in Iteration III, defined and structured by typical architectures, standards or/and realized by COTS components).


	WISA & Reference Architecture Deliverable ID: D4 B	Page : 50 of 53
		Version: 2.00 Date: 23 Oct 03
		Status : Proposal Confid. : Restricted

7.5 SUMMARY

The reference architecture currently consists of three architectural views on the conceptual level. The structural view uses layered style, and the layering is based on the taxonomy defined in chapter 4. A number of support service components relevant to wise pilots are also included into the structural view. The deployment view defines the deployment units of wireless service architectures and a reference deployment of a wireless service. These views can be used as the basis for the development of conceptual structure and deployment of a new wireless service, and help designers to analyze what support services are needed. For the Wise pilots the most important of these should probably be the service management and data management services.


The development view shows the services selected into service catalog as a part of reference architecture. This helps to analyze what available components could be used as such and which ones need to be adapted or developed? This also provides the starting point to estimate investment required for the development of a service product. Even if not used directly in pilots these components could provide valuable reference designs that can be used as the basis for designing architecture of components developed from scratch or improving designs in future iterations.

The need for more concrete reference models is an open issue for the next iteration of reference architecture. The problem with concrete views is the danger of defining yet another platform for a wireless service developer to choose from. Using the patten guidelines and components selected into handbook in the concrete architecture design partly solves this problem. On the other hand one or more good concrete architectural models could considerably help a new wireless service developer and help to design a service that can be easily ported to various wireless service platforms.


	WISA & Reference Architecture Deliverable ID: D4 B	Page : 51 of 53
		Version: 2.00 Date: 23 Oct 03
		Status : Proposal Confid. : Restricted

8. REFERENCES

- [1] Bass, L., Clement, P. and Kazman, R. Software Architecture in Practice. Addison-Wesley. 1998.
- [2] Buschmann, F., Meunier, R. and Rohnert, H., 1996. Pattern-oriented software architecture, a system of patterns. John Wiley & Sons.
- [3] 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.
- [4] 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.
- [5] 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)
- [6] 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.
- [7] Gnutella/Napster Comparison, URL <http://www.gnutellanews.com/information/comparison.shtml>.
- [8] 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.
- [9] Homayounfar, H., 2002. An advanced P2P architecture using autonomous agents. University of Guelph (Ontario, Canada). Master's Thesis. January 2002. 182 p.
- [10] Parameswaran, M.; Susarla, A.; Whinston, A.B., 2001. P2P Networking: An Information-Sharing Alternative. IEEE Computer, July 2001. Lerner, M., Vaneck, G., Vidovic, N., Vrsalovic, D., 2000. Middleware Networks- Concept, Design and Deployment of Internet Infrastructure. Boston Kluwer Academic Publishers. 375 p.
- [12] 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)
- [13] Buschmann, F. Building Software with Patterns, Proceedings of the Fourth European Conference on Pattern Languages of Programming and Computing, 1999, Bad Issee, Germany, 58p.
- [14] Giese, H., 2001. Design Pattern and Software Architecture: Software Architecture. Available on-line at: <http://www.uni-paderborn.de/cs/ag-schaefer/Lehre/Lehrveranstaltungen/Buschmann, F., Meunier,R., Rohnert, H., Sommerlad, P., Stal, M. 2000. Model-View-Controller. Available on-line at: http://vilalta1.tempdomainname.com/pages/PatronsDisseny/Pattern%20Model%20View%20Controller/Telemanagement Forum. On-line at: http://www.tmforum.org/>
- [17] 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
- [18] Fisher, G. E. Guide on Opens System Environment (OSE) Procurements. NIST Special Publication 500-220. Oct. 1994.
- [19] 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.
- [20] Dailey, H. PIECING IT TOGETHER, New J2EE Patterns Catalog Helps Solve the J2EE Architecture Puzzle. <http://java.sun.com/features/2001/03/patterns.html>.
- [21] Ihme, T., Kalaoja, J., Kallio, P., Niemelä, E., Torchiano, M. WISA Handbook, Deliverable ID: D4 (Part D).
- [22] Hartman, R. Building on patterns. ADT may 2001. 9 p.
- [23] 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>

	WISA & Reference Architecture Deliverable ID: D4 B	Page : 52 of 53
		Version: 2.00 Date: 23 Oct 03
		Status : Proposal Confid. : Restricted

- [24] Gamma, E., Helm, R., Johnson, R. and Vlissides, J. 1995). Design patterns: Elements of Reusable Object-Oriented Software, New York, Addison-Wesley, 395p.
- [25] 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.
- [26] Borchers, J. 2001. A Pattern Approach to Integration Design, John Wiley.
- [27] 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.
- [28] 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.
- [29] Kon, F., Costa, F., Blair, G. and Campbell, R. 2002. The Case for Reflective Middleware, Communications of the ACM, 45, 6, pp. 33-38.
- [30] Agha, G. (2002). Adaptive Middleware, Communications of the ACM, 45, 6, pp. 31-32.
- [31] Schmidt, D. 2002. Middleware for real-time and embedded systems, Communications of the ACM, 45, 6, pp. 43-48.
- [32] Raatikainen, K. Middleware in Mobile World, OT Land, LogOn Technology Transfer GmbH, June, 2003.
- [33] 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/>
- [34] 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.
- [35] 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.
- [36] 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. 439-494.
- [37] 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/>
- [38] 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.
- [39] 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.
- [40] 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.
- [41] 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.

	WISA & Reference Architecture Deliverable ID: D4 B	Page : 53 of 53
		Version: 2.00 Date: 23 Oct 03
		Status : Proposal Confid. : Restricted

- [42] 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.
- [43] 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.
- [44] 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
- [45] 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.
- [46] Alur, D., Crupi, J., Malks, D. 2001. Core J2EE Patterns: Best Practices and Design Strategies. Prentice Hall. 496 p.
- [47] 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
- [48] 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.
- [49] 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
- [50] 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.
- [51] Lutz, J. 2000. EAI Architecture Patterns. EAI Journal, March 2000. Pp. 64-73.
- [52] 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.
- [53] 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).
- [54] Neil, M. 2001. Turning to OO Platforms During Mobile Phone Design. Available on-line at: <http://www.commsdesign.com/story/OEG20011120S0075>
- [55] 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.
- [56] The OOPSLA 2001 workshop 'Towards Patterns and Pattern Languages for OO Distributed Real-time and Embedded Systems', <http://www.cs.wustl.edu/~mk1/RealTimePatterns/OOPSLA2001/>
- [57] PLOP 2002, Focus Topic "Patterns and Pattern Languages for Distributed Real-Time and Embedded Systems", <http://jerry.cs.uiuc.edu/%7Eplop/plop2002/>
- [58] The OOPSLA 2002 workshop 'Patterns in Distributed Real-time and Embedded Systems', <http://www.cs.wustl.edu/~mk1/RealTimePatterns/OOPSLA2002/>