ESPOO 2002 VTT PUBLICATIONS 476

Markus Moilanen

Middleware for Virtual Home Environments

Approaching the Architecture

VTT PUBLICATIONS 476

Middleware for Virtual Home Environments

Approaching the Architecture

Markus Moilanen

VTT Electronics

ISBN 951–38–6003-5 (soft back ed.)
ISSN 1235–0621 (soft back ed.)

ISBN 951–38–6004-3 (URL: http://www.inf.vtt.fi/pdf/)
ISSN 1455–0849 (URL: http://www.inf.vtt.fi/pdf/)

JULKAISIJA – UTGIVARE – PUBLISHER

VTT, Vuorimiehentie 5, PL 2000, 02044 VTT
puh. vaihde (09) 4561, faksi (09) 456 4374

VTT, Bergsmansvägen 5, PB 2000, 02044 VTT
tel. växel (09) 4561, fax (09) 456 4374

VTT Technical Research Centre of Finland, Vuorimiehentie 5, P.O.Box 2000, FIN–02044 VTT, Finland
phone internat. + 358 9 4561, fax + 358 9 456 4374

VTT Elektroniikka, Kaitoväylä 1, PL 1100, 90571 OULU
puh. vaihde (08) 551 2111, faksi (08) 551 2320

VTT Elektronik, Kaitoväylä 1, PB 1100, 90571 ULEÅBORG
tel. växel (08) 551 2111, fax (08) 551 2320

VTT Electronics, Kaitoväylä 1, P.O.Box 1100, FIN–90571 OULU, Finland
phone internat. + 358 8 551 2111, fax + 358 8 551 2320

Technical editing Maini Manninen

Otamedia Oy, Espoo 2002

Moilanen, Markus. Middleware for Virtual Home Environments. Approaching the Architecture. Espoo 2002. VTT Publications 476. 115 p. + app. 46 p.

Keywords software systems, software layers and subsystems, conceptual architecture, object-oriented analysis OOA, application programming interfaces, APIs, unified modelling language UML, user interface mark-up language UIML, generic user interface, remote service development, serverless service provisioning

Abstract

Virtual Home Environments (VHE) is the concept that networks supporting mobile users should provide them with the same computing environment on the road as they are used to having in their home or corporate computing environment. Middleware for Virtual Home Environment (the VHE Middleware project) is one of the ITEA (Information Technology for European Advancement) projects. The goal of the project is to make European industry the leader in Middleware software technology for end-user terminals, with wireless connections and the corresponding infrastructure to enable VHE. The project partners are Nokia (FIN), Siemens (D), Fujitsu-Siemens (D), Orga Kartensysteme (D), Paderborn University (D), Philips (NL, B, D), Robert Bosch (D), and VTT Electronics (FIN). This publication documents the research and development of VHE Middleware carried out by VTT Electronics. In a scientific sense, a full case study evaluating an advanced method of identifying the conceptual architecture of a software system from its functional requirements is presented. The proposed method is based on the common knowledge of object-oriented analysis (OOA) methodology, which claims that every software system contains hierarchical structures that reflect its functional requirements. In OOA methodology, these structures and their hierarchies can be found by analysing and structuring the problem descriptions - the Use Case analysis. The advanced method here is how to seamlessly move from a use case model to a conceptual architecture model of the software system. User's scenarios and software prototypes of the VHE system are used as a case study. Using the proposed method, the VHE system's subsystems and layers, and their corresponding application programming interfaces (APIs), are found and the conceptual architecture of VHE Middleware drawn up. After this, the technical development of the case is extended to fully concrete the VHE Middleware architecture and its elements.

Preface

ITEA (Information Technology for European Advancement, http://www.itea-office.org/) is a European-wide research and development (R&D) programme seeking to stimulate and support the development of competencies in software technology as a benefit to European industry. This is to be achieved by improving the quality of embedded and distributed software, thereby enabling greater innovation and richness of function in complex, software-intensive systems. ITEA's work is organised in projects with a typical duration of two years. The programme is managed at the European level within the EUREKA framework for R&D (http://www.eureka.be/), and involves a great deal of co-operation between nations, industrialists and public research centres. Being part of EUREKA, ITEA interfaces with other EUREKA projects and with the Framework Programmes of the European Commission's IST (Information Society Technologies) programme (http://www.cordis.lu/ist/home.html). Middleware for Virtual Home Environments (the VHE Middleware project, http://www.vhe-middleware.org) is one of the ITEA projects (ITEA 99013). This publication deals with the VHE Middleware project. The project's work packages (WPs) are "WP1: System Requirements & Design", "WP2: VHE Middleware for Multi-Standard Terminals", "WP3: VHE Middleware for Smart Card Platform", "WP4: VHE Middleware for VHE User-interface", "WP5: Technology Integration, Validation, Tests, and Documentation" and "WP6: Dissemination and Standardisation". VTT Electronics, being one of the project members, participated in a set of the project's WPs. This publication contributes to the WPs as follows:

WP 1 - "System Requirements and Design". The scope of this WP is explained through its tasks, as follows:

- **Task 1.1 - "Technology Tracking and Platform/Device Selection"**. The contribution dealing with the abstract requirements of VHE in general is published in Appendix C.

- **Task 1.2 - "Scenario and Prototype Definition"**. This task is presented in this publication by republishing the three user scenarios of possible VHE services for end-users in Appendix A, and as the functional requirements of VHE and the base for their further analysis.

- **Task 1.3 - "System Modelling and Architectural Design".** The purpose of this task was to initialise the development of the VHE Middleware software architecture. The evaluations of the architectural modelling mechanism and specification techniques were also to be included. The conceptual design of the VHE Middleware architecture presented in this publication is intended to fulfil both purposes.

- **Task 1.4 - "Component Modelling and API-design".** The purpose of this task was to achieve agreement on a common description format for the interfaces and to assign responsibilities aspects of VHE. The development of the VHE Middleware APIs presented in this publication is intended to fulfil this purpose.

WP 4 - "VHE Middleware for VHE User Interface". The development of a generic, scalable, multi-modal user interface concept for making connections that can be implemented with voice input/output in combination with the small monochrome displays of telephones right up to the large colour displays on TV sets and computer monitors. Because the VTT Electronics contribution to this WP is used as a case study here, it is reproduced in Appendix B.

WP 6 - "Dissemination and Standardisation". This WP deals with the promotion and standardisation of VHE Middleware technology. This text will contribute to this WP as a whole.

Personally:

Personally, I have been working as a participator in the VHE Middleware project for over two years. My work was in the R&D of the VHE Middleware's architecture. Although the subject is amenable to academic dissemination in many forms, this is not entered into in this publication.

Several people have contributed to both the publication and the project. All of these people will be acknowledged later in this publication. For further information, please see the Acknowledgements chapter. However, I would like to emphasise the role of Mr. Hannu Rytilä, Mr. Johan Plomp, and Mr. Tapani Rantakokko – many thanks to them for their support.

Furthermore, I would like to thank all European project partners.

Special thanks belongs to the reviewer Dr. Wolfgang Müller (e-mail: Wolfgang.Mueller@c-lab.de) from Paderborn University for his suggestions and comments which I have carefully taken into account.

Finally, I also wish to express my thanks to the personnel of VTT organisation and Tekes (the National Technology Agency of Finland) for making the work possible by the support and funding.

Oulu, Finland, 30 November, 2002

Markus Moilanen, M.Sc. Personal information and contacts at http://www.iki.fi/markus.moilanen

Contents

List of Abbreviations

3D	Three Dimensional
3GPP	The 3rd Generation Partnership Project
AM	Architecture Model
API	Application Programming Interface
B	Belgium
CORBA	Common Object Request Broker Architecture
D	Federal Republic of Germany
DM	Dynamic Model
EDIT	Enterprise and Desktop Integration Technologies
ETSI	European Telecommunications Standards Institute
FIN	Finland
FTP	File Transfer Protocol
GUID	Globally Unique Identifier
GPRS	General Packet Radio Service
GRIDS	Grid Research Integration Deployment and Support
HLR	Home Location Register
HTML	Hyper-Text Mark-up Language

HTTP	Hyper-Text Transfer Protocol
IDL	Interface Definition Language, Interface Description Language
IETF	Internet Engineering Task Force
IIOP	Internet Inter-ORB Protocol
IP	Internet Protocol
IR	Infra-Red
ISO	International Standardisation Organisation
ITEA	Information Technology for European Advancement
ITU	ISO Telecommunication Union
JMS	Java Messaging Service
LON	Local Operating Network
MExE	Mobile Execution Environment
MVC	Model-View-Controller
NA	Not Available
NFR	Non-Functional Requirements
NL	Netherlands
NMI	NSF Middleware Initiative
NSF	National Science Foundation
OCL	Object Constraint Language

OMA	Object Management Architecture
OMG	Object Management Group
OO	Object-Oriented
OOA	Object-Oriented Analysis
OS	Operating System
OSGi	Open Service Gateway initiative
PCC	Project Co-ordination Committee
PDA	Personal Digital Assistant
PLA	Product Line Architecture
PSE	Personal Service Environment
PKI	Public-Key Infrastructure
QoS	Quality of Service
RAD	Rapid Application Development
RMI	Remote Method Invocation
SCF	Service Capability Features
SCS	Service Capability Servers
SM	Static Model
TCP	Transfer Control Protocol
TS	Technical Specification

UDP	Unreliable Datagram Protocol
UI	User Interface
UID	Unique Identifier
UIML	User Interface Mark-up Language
UMTS	Universal Mobile Telecommunication System
UML	Unified Modelling Language
URI	Uniform Resource Identifier
VCR	Video Cassette Recorder
VHE	Virtual Home Environments
VR	Virtual Reality
VXML	Voice Extensible Mark-up Language
WAP	Wireless Application Protocol
WLAN	Wireless Local Area Network
WML	Wireless Mark-up Language
WP	Work Package
WWW	World Wide Web
XML	Extensible Mark-up Language

1. Introduction

Mobile appliances and nomadic ubiquitous computing are beginning to play an important role in current developments of information technology. Virtual Home Environments (VHE) is the concept that networks supporting mobile users should provide them with the same computing environment on the road as they are used to having in their home or corporate computing environment. The home services provided at a user's home should still be usable outside the home - users should be able to take advantage of their home environments when they are away from their homes. The VHE Middleware project [1] aims to define the middleware software technologies to be used in the different devices for establishing VHE. VHE will allow users to retain and personalise the home services under their authority wherever they are, and use them at any time, in both wireless and wired environments. A core component of VHE will be a generic connection service, which will enable residential or business users to contact back-office or home services in an ad-hoc fashion, independent of the environment (wired or wireless), and the access device – such as mobile phone, PDA (Personal Digital Assistant), desktop computer, etc. Beyond the technical level, it is within the scope of the project to identify those VHE services that attract end users and potential providers of the services. In fact, the VHE Middleware project was started by developing several descriptions of these VHE-service candidates. After that phase, the project was continued by developing several prototype systems reflecting users' scenarios in the selected VHE-service candidates.

The purpose of this text is twofold: to present the scientific results of the software research accomplished in the VHE Middleware project, and to document the contemporary technical developments of the VHE Middleware.

In the scientific sense of contributing to computational science, this publication will evaluate an advanced method of identifying a conceptual architecture and APIs of the software system from its functional requirements. Users' scenarios of the VHE-system, and their corresponding software prototypes, will be used as a case study. Using the proposed method, the VHE-system's subsystems, and their software layers and corresponding APIs (Application User Interfaces), will be found and, consequently, the conceptual architecture will be drawn up.

In the sense of technical development, this text will establish preliminary requirements for VHE Middleware. The requirements will be expressed as a middleware architecture concept and the preliminary API definitions of essential middleware components concretising the architecture. Furthermore, the documentation of the technical development will include the requirements and technical specification of a prototype system of VHE developed by VTT Electronics.

1.1 Existing VHE and Middleware Architectures

There are several ongoing research projects established to develop VHE technology. The VHE Middleware project is the first one that was intended to cover the issue of VHE Middleware technology. Several projects developing middleware technology for systems other than VHE certainly exist. As a short introduction to the issues of VHE and Middleware, a few examples from these categories will be presented in this section.

1.1.1 Object Management Architecture

CORBA (Common Object Request Broker Architecture) is OMG's (Object Management Group) open, vendor-independent architecture and infrastructure that computer applications use to work together over networks commonly identified as middleware architecture. Using the standard protocol of IIOP (Internet Inter-ORB Protocol), a CORBA-based program from any vendor, on almost any computer, operating system, programming language and network, can inter-operate with another CORBA-based program from the same or another vendor.

According to OMG, middleware architecture is the glue that holds enterprise applications together. The OMG's OMA (Object Management Architecture) [2] middleware architecture categorises objects into four categories: CORBA services, CORBA facilities, CORBA domain objects, and Application objects. Figure 1 presents the classic OMA architecture diagram. Applications, even if they perform totally different business tasks, share a lot of common functionality. For example, objects notify other objects when something

happens, object instances are created and destroyed, new objects' references are passed around, and the operation must be made secure and transactional. Beyond this, applications within a business domain - for example, the transportation business, banking business, etc. - share even more functionality. The OMA architecture abstracts out this common functionality from the CORBA applications into a set of standard objects that perform clearly defined functions accessed through standardised OMG IDL interfaces (Interface Definition Language). OMG standardises the IDL interfaces and specifies what the objects should do. Software vendors should then implement and sell implementations of these objects that perform the specified services.

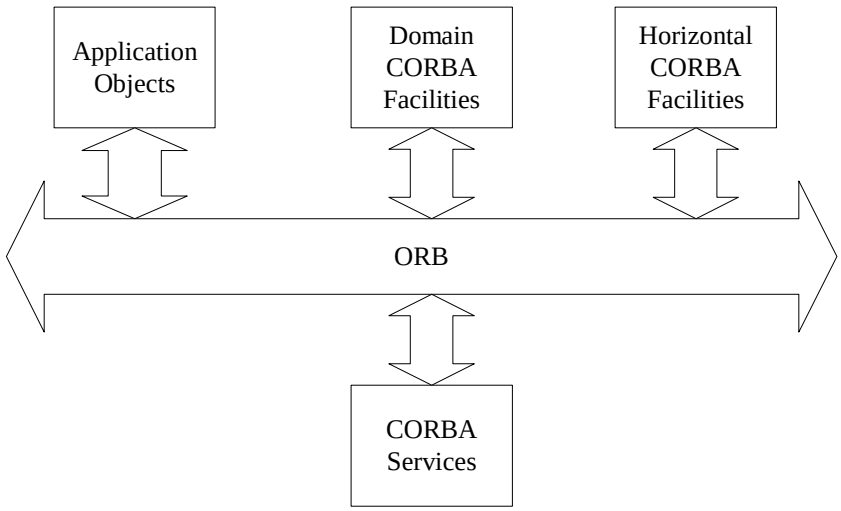

Figure 1. The classic OMA middleware architecture diagram.

Here is a view of each of the four parts of the OMA architecture (see Figure 1 above):

- The **CORBA Services** include the *Naming service*, *Object Trader service*, and the new *Persistent State service*.

- The **Application Objects** are at the topmost part of the OMA hierarchy. Since they are typically customised for an individual application and do not need standardisation, this category identifies objects that are not affected by the OMG standardisation efforts.

- The **Domain CORBA Facilities** are where the most work happens in OMG. IDL is a way of defining standard interfaces for standard objects that every company in an industry can share.

- The **Horizontal CORBA Facilities** sit between the CORBA Services and the Application Objects. Unlike the Domain CORBA Facilities, these facilities are potentially useful across business domains. There are only four horizontal CORBA facilities: the Printing facility, the Secure Time facility, the Internationalisation facility, and the Mobile Agent facility.

- The role of the ORB (**Object Request Broker**) is to enable a request to be carried out in a heterogeneous distributed environment. Clients can issue a request for objects, send the request to the appropriate objects, and prepare the objects to receive and process the request and return the results back to the clients. The ORB, therefore, implements a level of *distribution transparency*. The distribution transparency is defined in the ISO/ITU-T (International Standardisation Organisation / ISO Telecommunication Unit) standard 10746 [3] as a system's ability to mask out the heterogeneity and networked nature of distributed computing - the programming of distributed applications will become as easy as the programming of local applications.

1.1.2 Open Service Architecture

The 3GPP (The 3rd Generation Partnership Project) has standardisation activities to support the VHE concept as a part of their UMTS standardisation (Universal Mobile Telecommunications System). This standard is documented in the ETSI TS (European Telecommunications Standards Institute Technical Specification) 123 127 V3.0.0 (2000-3) [4], where VHE is defined as a concept for personal service environment (PSE) portability across network boundaries and between terminals. The concept of VHE is such that users are consistently presented with the same personalised features, user interface (UI) customisation and services in whichever network and whichever terminal, wherever the user may be located.

Open Service Architecture (OSA) defines an architecture that enables operator and third-party applications to make use of network functionality through an

open standardised interface (OSA Interface). OSA provides the glue between the applications and service capabilities provided by the network. By this, applications become independent of the underlying network technology - a typical feature of the existing middleware architectures. The applications constitute the top level of OSA. This level is connected to the Service Capability Servers (SCSs) via an OSA Interface. The SCSs map the OSA Interface Classes onto the underlying protocols and hide the network complexity from the applications. PSE describes how the user wishes to manage and interact with the communication services, which is a combination of a list of subscribed services, service preferences and terminal interface preferences. PSE also encompasses the user management of multiple subscriptions, e.g. business and private, multiple terminal types and location preferences. PSE is defined in terms of two kinds of user profiles: 1) the *user interface profile* and 2) the *user services profile*. The network functionality offered to applications is defined as a set of Service Capability Features (SCFs) in the OSA Interface. The SCFs are supported by different SCSs. The aim of OSA is to provide an *extendible* and *scalable* architecture that allows the inclusion of new SCFs and the SCSs in future releases of UMTS with a minimum impact on the applications using the OSA Interface. The SCSs serve as gateways between the network entities and the applications. The OSA API is based on the lower layers using mainstream information technologies and protocols. The middleware (e.g. CORBA) and lower layer protocols should provide security mechanisms to encrypt data.

The OSA-architecture is illustrated in Figure 2. The architecture consists of three parts:

- **Applications** implemented in one or more Application Servers.

- **Framework,** providing applications with basic mechanisms that enable them to make use of the service capabilities in the network. Examples of framework SCFs are *Authentication* and *Discovery*.

- **SCS**s, providing the applications with SCFs. SCFs are abstractions from the underlying network functionality. Examples of SCFs offered by SCSs are *Call Control* and *User Location*. The SCFs are specified in terms of a number of interface classes and their methods. The interface classes are divided into two groups: 1) the *framework interface classes*, describing the

methods on the framework, and 2) the *network interface classes*, describing the methods on SCSs.

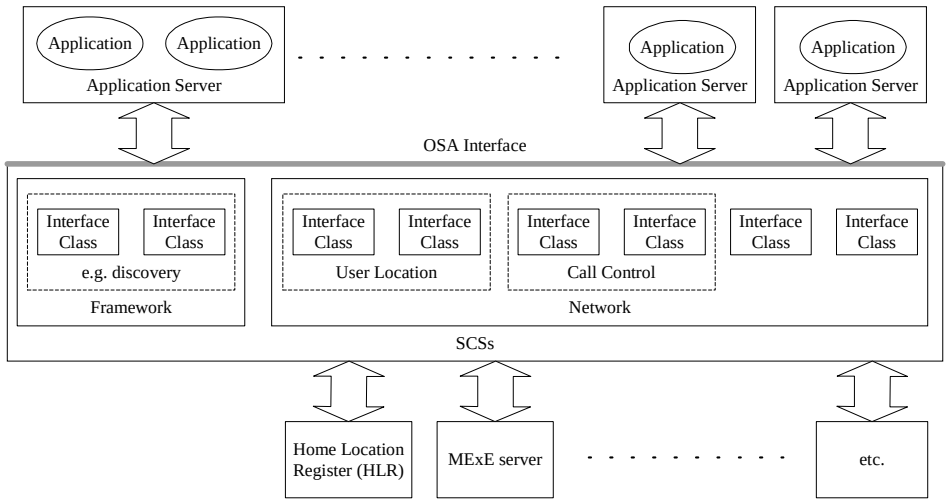

Figure 2. The OSA-architecture.

1.1.3 NSF Middleware Initiative

In late September 2001, the National Science Foundation (NSF) funded the creation of an NSF Middleware Initiative (NMI), (NSF02028) [5], to help scientists and researchers use the Internet to effectively share instruments, laboratories and data, and to collaborate with their colleagues. According to NMI, Middleware is software that connects two or more otherwise separate applications across the Internet. NMI will create and deploy advanced network services for simplifying access to diverse Internet resources. The initiative consists of two teams: the GRIDS (Grid Research Integration Deployment and Support) centre and the Enterprise and Desktop Integration Technologies (EDIT) consortium. The activities will facilitate the sharing of unique scientific resources such as telescopes, supercomputing systems or linear accelerators, as well as common resources such as databases, directories or calendars. The NMI technologies for user authentication and resource discovery could let students access national resources, including up-to-the-minute data and real-time

instrumentation. One important emphasis is to explore ways in which the Grid computing can be integrated with enterprise computing on university campuses.

“Internet2” is a project under the EDIT team of NMI and lists the next five services as being central of the Middleware as a whole, calling them the “Core Middleware”:

- **Identifiers** - A set of computer-readable codes that uniquely specify a subject.

- **Authentication** - The process of a subject electronically, establishing that it is, in fact, the subject associated with a particular identity.

- **Authorisation** - Those permissions and workflow engines that drive transaction handling, administrative applications and automation of business processes.

- **Directories** - Central repositories that hold information and data associated with identities. These repositories are accessed by people and by applications - for example, to get information, customise generic environments to individual preferences, and route mail and documents.

- **Security** - Certificates and public-key infrastructures (PKI) are related to the previous four core middleware services in several important ways.

Furthermore, according to Internet2, in addition to the Core Middleware there are numerous services that applications would like to have provided for them, rather than having to perform these functions themselves. These are called the ”Upper Middleware” and they can be grouped into three categories, as described below:

- **Business Application Middleware** – Typically, businesses are seeking extended tool sets with which to construct modular accounting or transactional applications. Four services are frequently mentioned in this category: *object resource brokering,* to find business data in distributed environments; *message handling,* both asynchronous and synchronously

between processes; *transaction monitoring,* to perform audit functions; and *application gateways* like electronic mail service (e-mail).

- **Research Application Middleware** - Complex networked computing environments are being developed to support the acquisition, processing and management of scientific data in unique situations where high-end resources are required. Network storage systems and specialised scientific instruments are being connected together into fabrics to address particular research agendas. Managing this environment requires a number of advanced services, such as *co-scheduling* of networked resources, *bandwidth brokers, library synchronisation,* and *database discovery.*

- **Ubiquitous Computing Tools** - To achieve "anywhere, anytime, anyhow" computing, users will need to be presented with a consistent, customised interface while employing a variety of devices from a variety of locations. While such *mobility, portability and ubiquity* will depend heavily on authentication and directory services, additional protocols and APIs need to be established. For example, standard sets of data will need to be moved frequently and securely between devices and centralised directories. Mechanisms are also needed to change attribute preferences, depending on the characteristics of the device being used.

1.1.4 Open Services Gateway Initiative

The Open Services Gateway Initiative (OSGi) [6] was founded in March 1999. Its mission is to create open specifications for the network delivery of managed services to local networks and devices. With over 80 member companies today, OSGi has a good chance to become the leading standard for the next-generation Internet services to homes, cars, small offices and other environments. The OSGi service platform specification delivers an open, common architecture for service providers, developers, software vendors, gateway operators and equipment vendors to develop, deploy and manage services in a co-ordinated fashion. It enables a new category of smart devices due to its flexible and managed deployment of services. The primary targets for the OSGi specifications are set top boxes, service gateways, consumer electronics, PC's, industrial computers, cars and more. These OSGi enabled devices will enable

service providers to deliver differentiated and value added services over their networks. Release 1.0 of the OSGi specification contained a specification for a service framework. This framework provides an execution environment for electronically downloadable services, called bundles. Deployed bundles are executed inside that framework and find a well-defined and protected environment. This environment includes a Java runtime and adds *life cycle management*, *persistent data storage*, v*ersion management* and a *service registry*. Services are Java objects implementing a concisely defined interface. The OSGi framework registry is used to exchange services between bundles in a secure and controlled manner. Through this registry, bundles may provide services to other bundles as well as use services from other bundles. The registry is fully security protected, allowing the operator full control over the platform. Release 1.0 specification included the Framework and three basic service specifications: *logging*, *a web server* and *device access*.

The OSGi Release 2 improves and extends the existing APIs. Security is strengthened and it is now possible to let a special management bundle fully define and control the security aspects of a bundle, in real time. This release also includes a number of new service specifications.

OSGi Framework

The Framework forms the core of the OSGi Service Platform Specification. It provides a general-purpose, secure, managed Java framework that supports the deployment of extensible and downloadable service applications known as bundles. OSGi-compliant devices can download and install OSGi bundles, and remove them when they are no longer required. Installed bundles can register a number of services that can be shared with other bundles under strict control of the Framework. The Framework manages the installation and update of bundles in an OSGi environment in a dynamic and scalable fashion, and manages the dependencies between bundles and services. It provides the bundle developer with the resources necessary to take advantage of Java's platform independence and dynamic code-loading capability in order to easily develop, and deploy on a large scale, services for small-memory devices. Equally important, the Framework provides a concise and consistent programming model for Java bundle developers, simplifying the development and deployment of services by decoupling the service's specification (Java interface) from its implementations.

This model allows bundle developers to bind to services solely from their interface specification. The selection of a specific implementation, optimised for a specific need or from a specific vendor, can thus be deferred to runtime.

The Framework allows bundles to select an available implementation at runtime through the Framework service registry. Bundles register new services, receive notifications about the state of services, or look up existing services to adapt to the current capabilities of the device. This aspect of the Framework makes an installed bundle extensible after deployment: new bundles can be installed for added features, or existing bundles can be modified and updated without requiring the system to be restarted. The Framework provides mechanisms to support this paradigm, which aid the bundle developer with the practical aspects of writing extensible bundles. These mechanisms are designed to be simple so that developers can quickly achieve fluency with the programming model.

Bundles

The OSGi [6] framework consists of installable core System Bundle, a selectable set of optional Management Bundles, and the Service Bundles that the application's programmer can implement if necessary for a specific application domain. The OSGi System Bundle must be installed to enable the other Service Bundles to be installed. To be valid, a System Bundle, as every other service bundle in the OSGi environment, must implement the interface "Bundle".

The OSCI-architecture

The three key aspects of the OSGi mission are multiple services, wide area networks, and local networks and devices. An abstract view of the wide area network, local network, and services gateway is shown in Figure 3. OSGi concentrates on the complete end-to-end solutions architecture from remote service provider to local devices. Because the OSGi specification focuses on providing an open application layer and gateway interface, it complements and enhances virtually all current local networking standards and initiatives.

The central component of OSGi specification effort is the services gateway that functions as the platform for many communications based services. The services gateway can enable, consolidate, and manage voice, data, Internet, and

multimedia communications to and from the home, office and other locations. The services gateway can also function as an application server for a range of high value services such as energy management and control, safety and security services, health care monitoring services, device control and maintenance, electronic commerce services and more. The gateway provides a focal point for service providers to deliver services to client devices on the local network(s). An implementation of the OSGi gateway may link client devices on the local network(s), such as energy meters, smart appliances or information appliances, to external service providers.

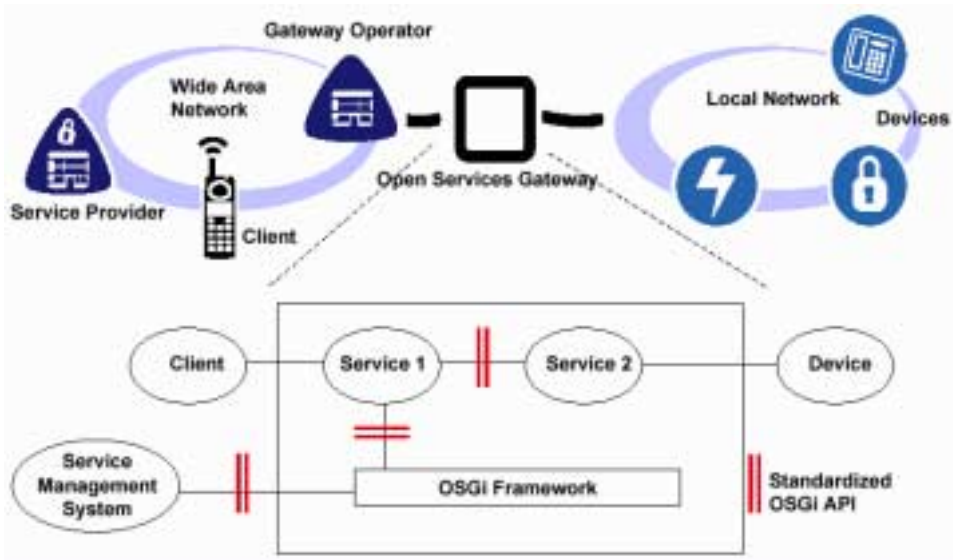

Figure 3. The OSGi-architecture.

1.1.5 Summary

As a summary of the previous section, it can be said that:

- **Middleware is a layer** (or layers) of software between the network and the applications.

- **Middleware provides compatibility** between heterogeneous systems and a set of services - such as location, profiling, identification, authentication,

authorisation, directories, security, etc. - to assist the interoperability of the systems.

- **Middleware is expressed as a collection of documents** that describe how the different systems can get along. In this case, the system itself must implement the compatibility issues described in the documentation.

- **The documentation can be assisted by a set of Middleware components**. Some components can provide environmental services - e.g. name service, directory services, file service, profiling service, etc. - and some can be used in constructing the services for the end users.

- **Middleware insulates application developers from having to understand the complexities of the computing environment**, such as network protocols. Application programmers can consider Middleware as a black box, where understanding the details of what happens inside is not required.

It can also be said that:

- **Middleware architecture describes how the services and their interactions should be built**. Respectively, the *inner-service* and *inter-service* types of architectures will exist. The inner-service architectures can be described by the hierarchies of subsystems and software layers. In the case of inter-service architecture, the question is about the communication and co-operation and their ways - the distribution transparency is an essential concept of the inter-service architectures, and it must be provided by the Middleware.

- **Middleware architecture is qualified according to the specific criteria of how the architecture fulfils its purpose** - a set of *quality attributes* is discussed as follows:

    - Middleware architecture must support the different ways of realising the necessary functioning for the different purposes and environments, and in using the different approaches, methods and tools - the Middleware architecture should be *flexible*.

- From the point of view of the VHE Middleware project, the idea of the NSI Middleware initiative looks most promising - the idea of Core Middleware which is extendable by the Upper Middleware. The extensions may support the realisations of the different service types that are the most common for the end users. At the same time, the Middleware architecture must also leave space to develop the support for future systems - the Middleware architecture should be *extendible*.

- It is not reasonable to assume that all the services associated with the Core Middleware could be supported by every realisation of VHE - a firm set of core services to be supported by VHE Middleware cannot be listed. In the mobile world, the services should be capable of being fitted to the system with the minimal processing power of cell-phones to full mainframe computers - the VHE Middleware architecture should be *scalable.*

1.2 VHE Use Cases

According to the VHE Middleware Consortium [2], successful development of software systems depends on the quality of the requirements engineering process. Use cases and user scenarios were considered promising vehicles for eliciting, specifying and validating requirements. Consequently, the use cases have been chosen as a first-phase tool in the development process. One major goal of the project is to elaborate VHE architecture. This section will contain a preview of a collection of user scenarios to enable this. The project partners have collected these scenarios as a first step to form a clearer common understanding concerning the benefit and usage of VHE architecture. The use case modelling technique was chosen for use in the VHE Middleware project to define the fundamental structure of VHE applications and the necessary VHE architecture to realise such applications. The VHE Middleware project has recommended a special template for use in describing the use cases in a uniform way, namely that developed by Cockburn et al. [7]. For examples using this template, see Appendix A. Twelve scenarios of candidate VHE services considered attractive to end users, called the “VHE Use Cases”, were finally accepted by the VHE Middleware project after the evaluation. Table 1 gives an overview of all the accepted VHE Use Cases:

Table 1. Overview of the accepted VHE Use Cases.

Scenario	*Short description*	*Presented by*
Use case - Garage Door Scenario	Describes how an end user approaching a garage door, opens/closes the door with a mobile device - e.g. a cell phone	Siemens
Use case - Movie Scenario	Describes how a user can play a movie from a VCR or DVD player on a TV set at home. The movie, VCR or DVD player and TV set is selected by a mobile device	Siemens
Use case - Generic UI	Describes how a home service can support the presentation of its User Interface (UI), and how any access device can deploy the same UI because of the UI's general format	VTT Electronics
Use case - Remote Service Development	Describes how the services can be developed and maintained remotely on behalf of the user, and how they can be used without the user's intervention, with the exception of "plug in the device and then use its services" - i.e. "Plug & Play"	VTT Electronics
Use case - Serverless Service Provisioning	Describes how a set of home appliances, brought together and switched on/off occasionally, can build up their services in an ad hoc way, and how these services can be made selectable and usable for a user	VTT Electronics
Use case - Move Broadcast	Describes how the VHE system pauses a running broadcast while a user is moving to another room	Paderborn University
Use case - Re-targeting Display	Describes how the VHE system redirects the display from one device to another (generic use case)	Paderborn University
Use case - VHE Pre-sets	Describes how the VHE system ensures that the pre-sets of the car radio store the same radio stations as the corresponding device at home (HiFi Tuner or PC with DAB Module)	Bosch
The table will continue		

Continuing Table 1		
Scenario	*Short description*	*Presented by*
Use case - VHE Play List	Describes how the VHE system ensures that the categories and filenames concerning the MPEG 3 content in the car is presented to the user in the same structure as on his PC at home/set-top box.	Bosch
Use case - VHE Home Theatre	Describes how the VHE system plays a clip in the home theatre	Nokia
Use case – Generic Security	Describes how the VHE system authenticates a user of mobile device and the remote server (client authentication / server authentication)	ORGA
Use case – Tennis Court Billing	Describes how the VHE system opens the entrance to a tennis court that belongs to the user's block of flats and bills the utilisation fee	ORGA

VTT Electronics is the author of three of the VHE Use Cases and their corresponding prototypes. These three VHE Use Cases are reproduced in Appendix A, and the corresponding prototype in Appendix B. The prototypes have provided valuable information on what would be the actual requirements for VHE Middleware. The prototypes have been demonstrated in public in the *International ITEA Workshop on Virtual Home Environments,* which was organised by the VHE Middleware project [8].

1.3 Approaches, Methods and Tools

To approach a conceptual architecture of VHE Middleware, an advanced method of identifying a conceptual architecture and APIs software system from its functional requirements will be proposed here. The proposed method is based on the common knowledge of the OOA methodology, which claims that every software system contains hierarchical structures that reflect its functional

requirements. In the OOA methodology, these structures and their hierarchy can be found by analysing and structuring the problem descriptions - the Use Case analysis is discussed. The proposed method here is how to seamlessly move from a use case model to a conceptual architecture model of the software system. To evaluate the applicability of the proposed method, the *Theory testing and Case-research* methodology [9, 10] will be deployed. In the first step, the proposed method will be refined. In the next step, a model of a problem in following the method will be developed, and, in the final step, the correctness of the model will be tested with a single case. The steps will be refined as follows:

Step 1. **Method introduction** - Analysing today's evolving information systems; the proposed method will be shown and explained thoroughly.

Step 2. **Creating a model of the problem** - Using the proposed method in structuring the problem, an analysis will be applied on a set of the VHE Use Cases and the corresponding models will be created.

Step 3. **Testing the correctness of proposed method with a single case** - To test the proposed method, the methodology of *Reverse Engineering* will be used here. The Reverse engineering method in software technology is to restructure and organise software code to reach its abstractions [11]. Consequently, the software code, which provides the middleware functioning of the prototypes, will be extracted and the Middleware components will be seen for first time.

In addition to these methods, the OOA methodology will be deployed. The results will be illustrated in using tables, UML-graphics (Unified Modelling Language) and the notation language of OCL (Object Constraint Language) [12].

1.4 Structure of the Document

The document is structured by chapters and appendices. The appendices present the source material for the problem and the case on which the further analysis is then applied in the main chapters as follows:

- **Appendix A** - This appendix introduces the three VHE Use Cases developed by VTT Electronics introduced in Table 1.

- **Appendix B** - This appendix introduces the prototype software presented by VTT Electronics. This prototype system is mainly built according to the "Use Case - GenericUI" introduced in Appendix A. However, some parts of the other two are also included. The prototype presents a UIML-based (User Interface Mark-up Language) [13] implementation of the GenericUI service concept - this is called the “UIML Service”.

- **Appendix C** - This appendix introduces the specification of the abstract requirements pre-set for the VHE-system and its Middleware by experts.

- **Evolving Information Systems** - In this chapter, to enter **Step 1** introduced in Chapter 1.3, the proposed method is developed. For this purpose, evolving information systems is analysed using the proposed method. The development of the Producer-Consumer paradigm to present, first, the decomposition model of the *manually-driven information system* and, second, its extension of the *automatic and semi-automatic information system* is presented. The approach to the use of the Producer-Consumer paradigm as a base for the decomposition model is presented in [14] for the first time.

- **Architecture Concept of VHE Middleware** and **APIs of VHE Middleware** - In these chapters, to enter **Step 2** introduced in Chapter 1.3, the VHE Middleware architecture’s concept and APIs are developed. The decomposition model of the semi-automatic information system, which was established in the previous chapter, is deployed in the analysis. The inputs are the VHE Use Cases in Appendix A - they are analysed thoroughly. The model presents the subsystems, their APIs and layering. The APIs are developed further in the second chapter. The output is a conceptual model of the VHE Middleware architecture and its related preliminary APIs.

- **Case Study** - In the chapter, a test of whether or not the use of the proposed method produces the correct architecture is conducted. By this, **Step 3** introduced in Chapter 1.3 is entered. To that end, the methodology of Reverse Engineering [11] is applied. The prototype software is analysed and

its software functions are fitted into the architecture model. The input is the decomposition model that was developed in the previous chapters and a prototype system reproduced in Appendix B. The output is the API definitions of the UIML Service written in Java.

- **Conclusions and Further Research** - As the purpose of this text was mentioned to be twofold, the conclusions is twofold as well; the conclusions in the sense of scientific research, and the conclusions in the sense of technical development of the VHE Middleware; where we have reached and where we are going to from there. The question of the applicability of the proposed method is evaluated, and the applicability and qualities of the technical solution is discussed. Furthermore, the issue of further R&D of VHE Middleware is looked into.

2. Evolving Information Systems

It can be assumed that future information systems will be based on earlier information systems that have been found the most successful in the sense of their prevalence - the information systems are evolving. Indisputably, the Internet and its service solutions, like WWW (World Wide Web) with HTTP (Hyper-text Transfer Protocol), SMTP (Simple Mail Transfer Protocol) or FTP (File Transfer Protocol), have been the most successful information system ever. In this chapter, to enter **Step 1** introduced in Chapter 1.3, an analysis of the evolving information systems will be carried out and, consequently, a decomposition model to provide a basis for further analysis of the VHE Use Cases will be created.

Functional requirements capture the intended behaviour of the system. This behaviour can be expressed as the services, tasks, or functions the system is required to perform. In the OOA methodology, the Use Case analysis [12] is used to decompose the problem and its functioning - a tool for functional decomposition is provided. Use cases comprise the presentations of actors and a set of functional descriptions, which interact. An actor is a user's role or anything that is able to provide independent events in the system to achieve its goal. The associated use case describes the functioning of the system by which the goal is to be achieved. Use cases capture who (actor) does what (interaction) with the system and for what purpose (goal), without dealing with the system internals. When the OO analysis is deployed, the process starts with identifying the actors and describing their corresponding system function. The essential system's objects will also be identified in the process. These identifications can be made by examining the system and its functional descriptions. This process is modelled by Use Case model graphics. Identifying the objects makes it possible to further model the system's behaviour with the services that the system should provide for the actors. These services are usually modelled by message sequences. In Section 1.3, we have proposed a new method for seamlessly moving from a use case model to a conceptual architecture model of the software system. In this chapter, the proposed method will be deployed in the development of the Producer-Consumer paradigm to present, first, the decomposition model of the *manually-driven information system*, and, second, its extension of the *automatic and semi-automatic information system*. In this way, the layers and APIs of the evolving information system will be exposed.

2.1 Manually-driven Information Systems

In applying the approach, a decomposition of the Producer-Consumer paradigm to express the manually-driven information systems will be presented.

The system that will be described here introduces a quite ordinary arrangement of how to produce and consume an information service - of how to produce and consume software in an Internet environment using, for example, an HTTP-service. These scenarios, which are in everyday use, define the *status quo* of where to start designing the new service concepts that the idea of VHE suggests. To start to present the results of the analysis, the actors are listed in Table 2. In the table, the first column of "Actor" presents the actors, the second column of "Super class" presents the generic class of entities a particular actor represents, and the third column of "Attributes" qualifies the actor's role - is it primary or secondary. The Primary actors are those the system is to be built for - usually the end users. The Secondary actors are those who assist the system to deliver the specific service for the primary actors. The "Note" column will clarify the issue.

Table 2. Actors of the manually-driven information system.

Actor	*Super class*	*Attributes*	*Note*
Platform Manufacturer	Industry	Secondary	A manufacturer of enabling technology
Service User	Human	Primary	A user of the system
Service Subscriber	Human	Primary	A human role to pay the bills
Service Provider	Human	Secondary	A human operator, happy with charging the subscribers
Service Developer	Human	Secondary	A developer of applications

Descriptions of the Use Cases will be presented in Table 3. The column "Use Case" names the use cases in question and the column "Descriptions" explains the activities between the actor and the use case.

Table 3. Use cases of the manually-driven information system.

Actor	*Use Case*	*Description*
Platform Manufacturer	Enables Service	Develops and manufactures techniques to enable the information systems to be built, e.g. a manufacturer of mobile phones, PDAs, etc.
Service Developer	Develops Service	Service Developer develops services by developing applications.
Service Provider	Provides Service	1. Service Provider configures a service by deciding upon the name and address information of the service to be provided. The provider's identity is known. 2. Service Provider initiates communication by preparing the network connection of the server. Communication is reserved according to the provider's identity, by which the accesses to the available communications channels are allocated. 3. Service Provider initialises the service on the server by installing the application. 4. Service Provider notifies the availability of the service to the potential service subscribers by registering the service.
Service User	Uses Service	1. Service User starts the client software. Access rights and personal settings of the user in using the service might be concerned. Identity of the user is known and will be deployed at the application level to guarantee personal security. 2. Service User uses the service. Because of the known identity, the user might be charged for the use of the service. 3. Service User closes the client software.
The table will continue		

Continuing Table 3		
Actor	*Use Case*	*Description*
Service Subscriber	Consumes Service	1. Service Subscriber configures a service by deciding upon the name of the service to be subscribed. The subscriber's identity is known. 2. Service Subscriber initiates communication by preparing the network connection. Communication channels are reserved and allocated according to the subscriber's identity. The identity is a key to access available communication channels. 3. Service Subscriber connects to the server and retrieves the service by downloading the service's client side software. The client side software is made available by the service provider. The known identity of the subscriber makes it possible for the Service Provider to charge for the software. 4. Service Subscriber performs the service by installing the client software for service users.

The analysis model is presented as UML graphics in Figure 4.

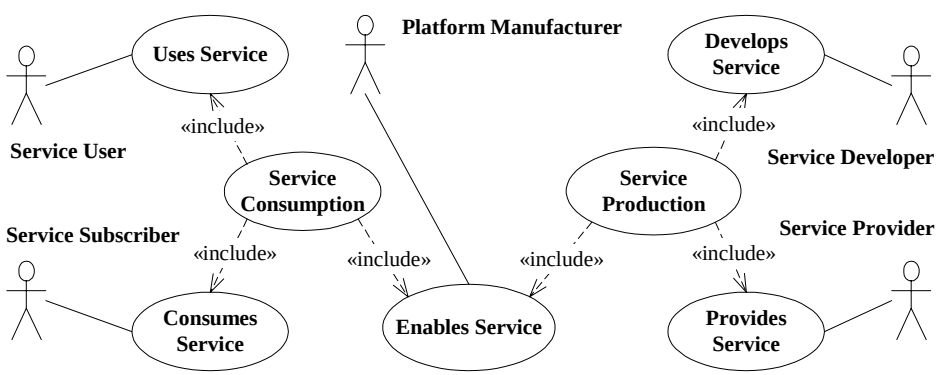

Figure 4. Actors and Use Cases of the manually-driven information system.

2.2 Automatic and Semi-automatic Information Systems

In the automatic information systems, machine-origin actors drive the system instead of individual users. The actors and use cases of manually-driven information systems will be overridden by the extended systems in order to provide and consume services automatically. However, given that the extended systems will perform functions differently to the old ones, some of the actor's roles in the overridden systems may remain. For example, billing functions may still be working as they have been working in the manually-driven information system - this makes the system “Semi-automatic”. Actually, being Semi-automatic means that the machine-origin actors only assist the system to fulfil the goals of the “Primary” ones, therefore this kind of assisting actor should be called the “Secondary” one. When such a system evolves, there is no limit to providing the billing functions automatically - for instance, in using the concept of e-purse (electronic purse), etc. Nevertheless, the individual users will still work manually in the future.

2.3 Elaboration of the Producer/Consumer-paradigm for Semi-automatic Information Systems

The Semi-automatic information system will extend the manually-driven one by performing some functions differently or by providing new functions. Commonly, the machine origin actors called the "Client" and "Server" will assist the functioning to take place, overriding the human actors - the *Client-Server-architectural pattern* will appear [15]. These kinds of machine-origin actors are called the "Daemon". In UML, the extension of a system can be modelled by presenting special *extension points*. In this case, the manually-driven information system will be extended by an automatic system. This is illustrated in Figure 5, where the extension point of "Supports Automatic Service" is presented. To make it possible to track the extension points when the analysis progresses with the other extensions in subsequent chapters, the extension points will be labelled with numbers, and this one will be labelled with the number 1. For further information on the concept of UML’s extension points, please consult the most recent UML manual [12]. By introducing the extension points with the evolving system, the vertical decomposition - i.e. layering of the system - will be supported. Every new layer will introduce a new set of definitions by which the

layer can be deployed - a new API, which will match a particular extension point, will be presented. Actually, *this is the most essential point of the proposed method*. The proposed method can be exploited when a new system is built on the previous one - only the extended functionality and its corresponding APIs will be required to be presented by the system that extends the previous one.

The results of an OOA on the Semi-automatic information system will be presented as follows: Table 4 introduces the actors and Table 5 presents the analysis of the system. Figure 5 clarifies the results using UML graphics.

In Table 4, the columns' titles will be explained as follows: the title "Description of Actor" clarifies the actor's role. The titles "Subclass" and "Super class" states the support for the development of the class hierarchy of the system. The others titles remain as explained earlier. The concept of Subclass and Super class is clarified in Figure 6.

Table 4. Actors of the Semi-automatic information system.

Description of Actor	*Actor*	*Subclass*	*Super class*	*Attributes*
A user of an automatic service	User	NA	Human	Primary
A developer of an automatic service	Service Developer	NA	Human	Secondary
A consumer of an automatic service	Client	VHE_Client	Daemon	Secondary
A provider of an automatic service	Server	VHE_Server	Daemon	Secondary

In Table 5, the titles of the columns not previously introduced in this text are as follows: the title "Description of Use Case" will explain the use case and the title "Use Case" will name the explained use case. The title "Included Use Cases" will give the names of the sub-use cases that are considered to be included in the main use case named in the column "Use Case" - this is according to the UML's «include» stereotyped relationships between the use cases [12]. The last one, "API", will give the name for the APIs considered to be appearing in the extended system.

Table 5. Functional decomposition and APIs of the Semi-automatic information system.

Description of Use case	*Use Case*	*Actor*	*Included Use Cases*	*API*
An automatic server can be switched On/Off	Provides Service	Server	Supports On/Off	"Supports Automatic Service"
Service Provider *configures a service*			Configures Service	
Service Provider *initiates communication*			Initiates/Un-initiates Communication	
Service Provider *initialises* the service on the server			Initiates/Un-initiates Service	
Service Provider notifies availability of the service			Registers/Un-registers Service	
Service Developer *develops services* by developing applications	Develops Service	Service Developer	Develops Applications	
An automatic client can be switched On/Off	Consumes Service	Client	Supports On/Off	
Service Subscriber *configures a service*			Configures Service	
Service Subscriber *initiates communication*			Initiates/Un-initiates Communication	
Service Subscriber gets connected to the server and *retrieves the service*			Retrieves Service	
Service Subscriber *performs the service*			Presents Service	
The table will continue				

Continuing Table 5				
Description of the use case	*Use Case*	*Actor*	*Included Use Cases*	*API*
Service User starts the client software	Uses Service	User	Starts service	"Supports Automatic Service"
Service User uses the service			Uses service	
Service User closes the client software			Closes service	

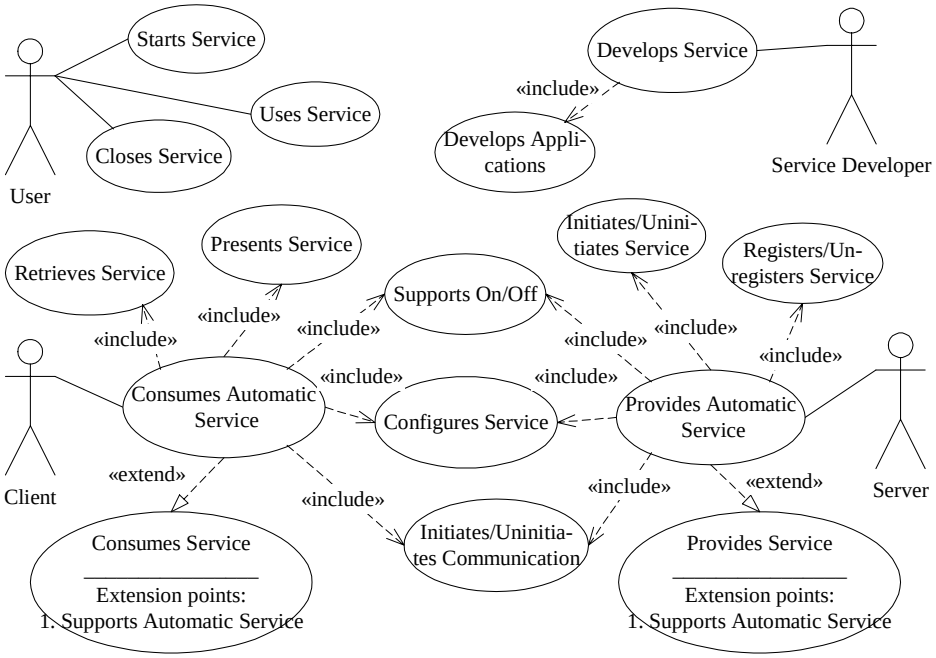

Figure 5. Functional decomposition and APIs of the Semi-automatic information system.

Figure 6 refines the concept of "Subclass" and "Super class" in the OO methodology used to express the hierarchy of the classes used throughout this text and Tables. In the figure, the example illustrates how the class "VHE_Class_2" inherits the "VHE_Class_1". Furthermore, it illustrates how the

“VHE_Class_2” - in addition to being a super class for the “VHE_Class_3” - is a subclass of the “VHE_Class_1”. In this way, large class hierarchies can be supported. This is congruent with the UML specifications, except that the term “super class” is written as “superclass” in UML (v.1.4).

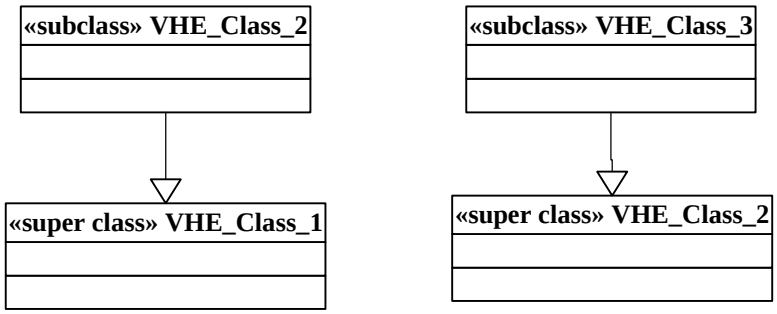

Figure 6. Concept of “Subclass” and “Super class” of the OO methodology used to express inheritance hierarchies of the classes in Tables and texts through this book.

2.4 Chapter Summary

In this chapter, the proposed method was introduced and the Producer/Consumer paradigm was developed in using the method. **Step 1** introduced in Chapter 1.3 was entered. The paradigm was developed according to the idea of the evolving information system presented in the chapter, and the result was a functional decomposition model of the Semi-automatic information system. In the analysis, the UML’s concept of extension point (expressing how a system might be extended) was deployed. Consequently, in the model, the layers and APIs of a software system were seen exposed. We may conclude that the proposed method is likely to work when we continue the analysis of the VHE system.

3. Architecture Concept of VHE Middleware

Software architecture is a high-level abstraction of a system, providing a forum with which to reach a full understanding and consensus over the system and amongst the stakeholders. It also makes it possible to apply early design decisions to the system, and, for that purpose, can be discussed from the viewpoint of the stakeholders. On the other hand, software architecture is a set of modelling elements that define the software system. Modelling elements makes it possible to employ the reusable assets of software architectures and provides the blueprints for implementing the system in different environments. Correspondingly, the requirements for software architecture should be considered in the two following categories: 1) *functional requirements* and 2) *non-functional requirements.*

In this chapter, to initialise the **Step 2** introduced in Chapter 1.3, a conceptual architecture of VHE Middleware will be developed. The requirements of the VHE system will be considered. The method developed in the previous chapter to decompose the problem descriptions to find a system's APIs will be deployed.

3.1 Functional Requirements

Appendix C introduces the a*bstract functional requirements* pre-set by experts in the VHE-system. These requirements were considered during the development of the functional requirements for the applications domains of VHE. The functional requirements are described by the VHE Use Cases. The VHE Use Cases to be considered in this text are reproduced in Appendix A, and they will now be analysed using the OOA methodology.

3.2 Non-functional Requirements

The Non-functional requirements (NFR) originate in the taking care of those system requirements that cannot be explicitly set just by considering the primary needs of the end users. There are other stakeholders in the industrially produced

software systems - for example, the software industry, software developers and hardware manufacturers. An exhaustive analysis over quality attributes is presented in [16]. In addition, [17] illuminates the background to the architecture analysis methodology in general. Appendix C will present the analysis of NFR provided by experts in VHE architecture. According to the analysis, NFRs can be categorised into two main groups: 1) Product Line Architecture (PLA)-related quality attributes, and 2) Domain-related quality attributes. However, in addition to these two groups, in Section 1.1 we have identified one more group of qualities, namely a group, which concerns the Middleware architectures. The three groups will be introduced next:

- **PLA-related quality attributes**. These attributes can be discerned when software is under development and production, but they are not discernible at runtime [18, 19]. The PLA-related quality attributes are a concern of the software industry and they mainly support the reusability of parts of the software production and system. Examples include *reusability, modifiability, portability, integrability, and testability*.

- **Domain-related related quality attributes**. These attributes can only be discernible at runtime, i.e. when the system is in use. These attributes are a concern of the software developers and hardware manufacturers in providing a well-working implementation of a system. Examples include *performance, security, availability, functionality, and usability*.

- **Middleware-related quality attributes**. This group is not normally listed with software systems; however, it is essential for a Middleware system. According to the short analysis presented in Section 1.1.5, the quality of the Middleware architecture concept is affected by three main attributes: *flexibility, extendibility and scalability*.

3.3 Elaboration of the VHE Use Cases

The VHE system will be derived from the Service Production - Service Consumption information system, as the analysis of evolving information systems in the previous chapter has shown. Consequently, the VHE Use Cases will extend the functionality of the Semi-automatic system that was presented in

Table 5 and Figure 5. Figure 7 illustrates the Main Use Case diagram of the VHE system, and the stakeholders and corresponding *business cases*. Presentation of the Business Cases is the highest level view on the system. Who needs the system and for what purpose will be expressed in this view. The figure shows how the three VHE Use Cases developed by VTT Electronics extend the Use Cases described with the Semi-automatic information system (see Figure 5).

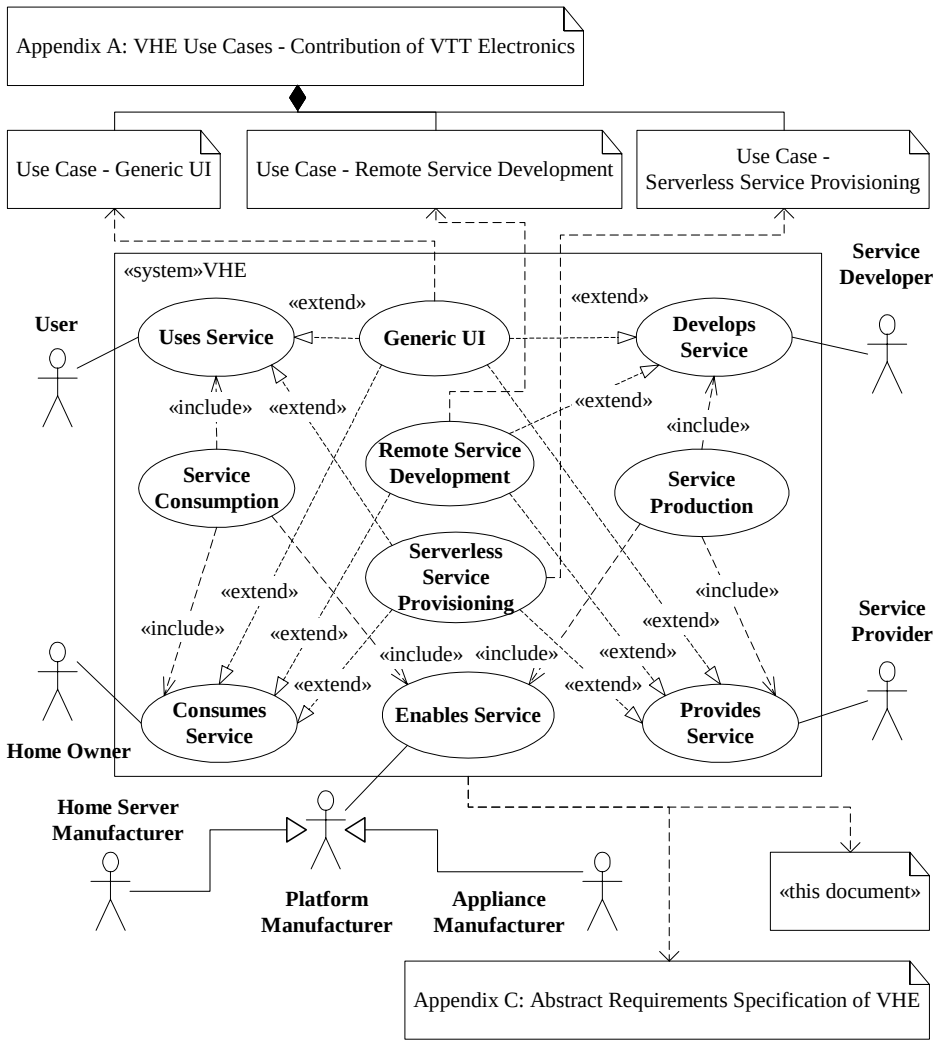

Figure 7. Service concepts of the VHE system.

To conclude the analysis here, the "GenericUI" service concept will extend the way in which services are developed, provided, consumed and used. The "Remote Service Development" service concept will only extend the way services are developed, provided and consumed; it will not extend the way the services are used - they will be used in exactly the same way as they were used before the extension. Furthermore, the “Serverless Service Provisioning” service concept will extend the way services are used, provided and consumed; it does not extend the way services are developed.

3.3.1 Elaboration of the "Use Case - GenericUI"

The concept of the GenericUI service covers the issue of how a home service can support the presentation of its UI, and how any access device can deploy the same UI because of the UI's general format. This scenario also supports the automatic initiation/restoration of home appliances that are switched on/off or plugged into a wall socket. In general, the Middleware architecture should enable the deployment of, for example, HTML, Jini's Proxies, UIML, Model-View-Controller (MVC) with Java Swings (the MVC concept is also deployed in the Symbian OS), Wireless Modelling Language (WML) used with the Wireless Application Protocol (WAP), Extensible Mark-up Language (XML), or even VoiceXML-based approaches. More information on the mentioned UI approaches concerning the intended system is presented in [20], [21], [22] and [23]. The target environments may vary from an embedded mobile environment to the legacy Internet. In addition, the implementation techniques may vary - e.g., from assembly languages to the high-end RAD-tools (Rapid Application Development). Furthermore, the VHE Middleware architecture must support the different home appliances in presenting their services by, for example, enabling the simplest possible device to present its services in the most minimalist way. Moreover, UI techniques under VR environments (Virtual Reality) are developing fast. VR gives enormous opportunities to provide even full-size three-dimensional (3D) monitoring and controlling capabilities integrated with clothes, shell-phones or even wristwatches. This support must be included in the VHE Middleware.

The results of the OOA on the "VHE Use Case - GenericUI" (see Appendix A) will be introduced in Table 6 and Table 7. Table 6 introduces the actors and

Table 7 summarises the analysis. The tables will also trace the original requirements definitions set by the column “Tracing the VHE Use Cases” in the VHE Use Cases. Please note that the abbreviation “NA” in the tables means “Not Available”, indicating that the original VHE Use Cases do not specify the particular issue or that the issue is not relevant at present. Figure 8 clarifies the results using UML graphics. In Figure 8, the system, which already presents "Supports Automatic Service" (see Figure 5), now has the three additional extension points introducing new APIs as follows:

2. "Supports Service Usage" - a client is able to retrieve a list of available services and to present it to a user. The system also supports user selection of a service.

3. "Supports GenericUI Service" - a system is able to support client devices in the device-specific way of presentation appliances' UIs, at least to present a view of but also to control and monitor a specific service that the appliance provides.

4. "Supports Home Services" - an appliance that already provides a specific service is also able to provide a description of its public data, to handle control events and to provide monitoring events to allow it to fulfil its responsibilities.

Item number one (missing on the list) is “1. Supports Automatic Service”, which was introduced in Section 2.3 and presented in Figure 5. Please also note that the listed support of "Supports Service Usage" and "Supports Home Service" are not bound to the concept of GenericUI - they will be deployed by other service concepts, as will be shown later.

Table 6. Actors of the "Use Case - Generic UI".

Tracing the VHE Use Cases	*Actor*	*Subclass*	*Super class*	*Attributes*
“The user of the system”	User	NA	Human	Primary
NA	Service Developer	NA	Human	Primary
NA	Home Owner	NA	Human	Primary
“Home server (also serving as gateway)”	UI Broker	VHE_UI_ Broker	Daemon	Secondary
“Appliance (may be more than one)”	Appliance	VHE_ Appliance	Daemon	Secondary
“Handheld device (client)”	Service Browser	VHE_Service_ Browser	Daemon	Secondary

Table 7. Functional decomposition and APIs of the "Use Case - Generic UI".

Tracing the VHE Use Cases	*Use Case*	*Actor*	*Included Use Cases*	*API*
"the home server will maintain a set of transcoding modules able to transcode the UI description format to other standardised formats"	Provides GenericUI Service	UI Broker	Maintains Transcodecs: - Retrieves and Updates Transcodec - Stores Transcodec - Installs/Unins-talls Transcodec	"Supports GenericUI Service"
"maintaining the UI descriptions of the controllable devices is the task of the home server" "the UI description must be retrieved from an external source" "checking for new versions" "storing UI description"			Maintains UI Descriptions: - Retrieves and Updates UI Descriptions - Stores UI Descriptions	
NA			Maintains Profiles	
NA			Generates Push Events	
NA			Handles Monitoring Events	
"the events must be passed to the device to be controlled"			Passes Events	
The table will continue				

Continuing Table 7				
Tracing the VHE Use Cases	*Use Case*	*Actor*	*Included Use Cases*	*API*
“a control device retrieves controllable devices' UI description from home server” “a control device maintains a cache of controllable devices' UIs”	Consumes GenericUI Service	Service Browser	Fetches and Updates UI	"Supports GenericUI Service"
“Presenting the UI”			Presents UI	
“a control device generates control events”			Generates Control Events	
NA			Handles Push Events	
“its list of controllable devices should be adjusted”			Retrieves Service Items	"Supports Service Usage"
“control device presents the updated list to the user”			Presents Service Items	
“when a controllable device is selected from the list”	Service Usage	User	Selects Service Item	
NA			Uses Service	
The table will continue				

Continuing Table 7				
Tracing the VHE Use Cases	*Use Case*	*Actor*	*Included Use Cases*	*API*
“the device receives the control events”	Provides Home Service	Appliance	Handles Control Events	"Supports Home Services"
NA			Generates Monitoring Events	
NA	Consumes Home Service	Home Owner	Manual operation: a home owner adds/removes a device at home	
NA	Develops Home Service	Service Developer	Manual operation: a service developer develops applications being informed about "Supports Home Service" API and its operations to be implemented	

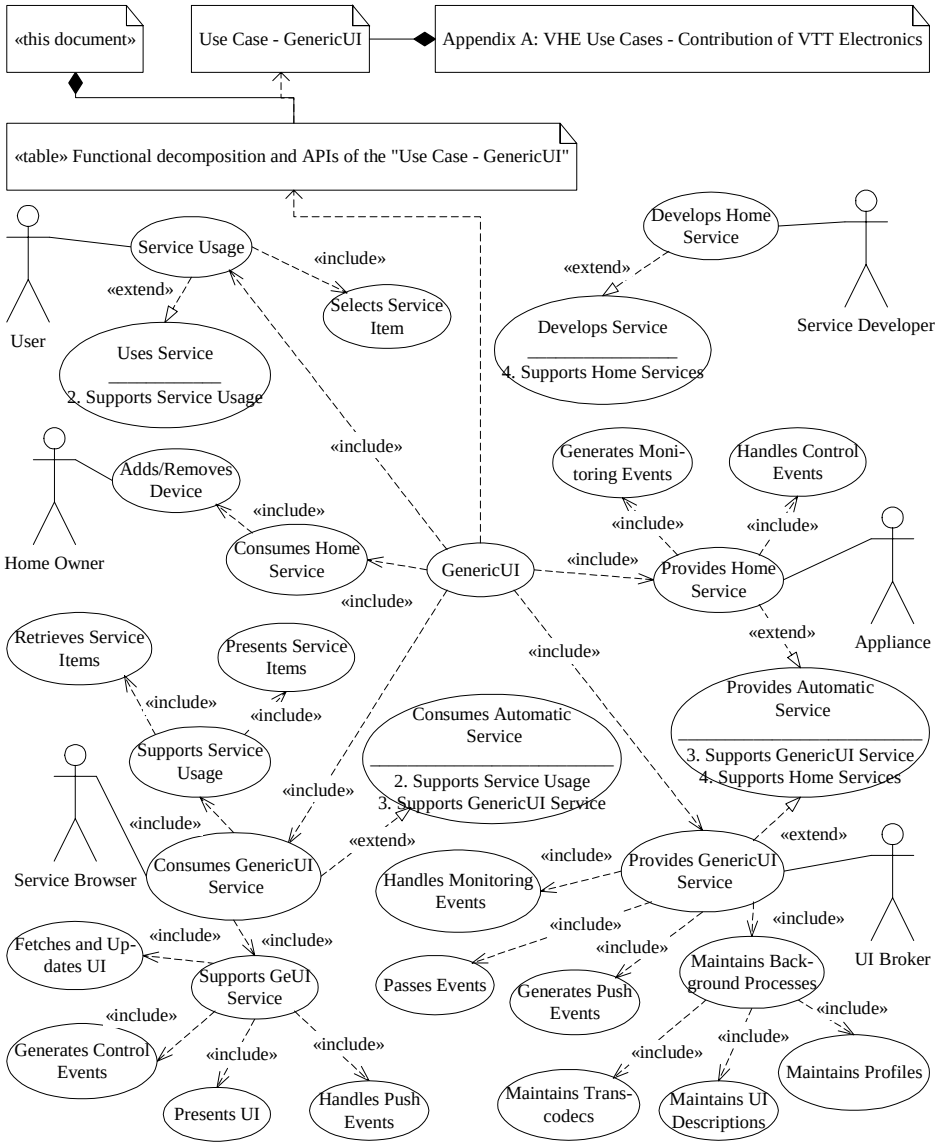

Figure 8. Functional decomposition and APIs of the "Use Case - GenericUI".

3.3.2 Elaboration of the "Use Case - Remote Development"

The concept of Remote Service Development covers the issue of how home services can be developed and maintained remotely on behalf of the user. It also describes how the home services can be used without the user's intervention,

with the exception of "plug in the device and then use its services" - i.e. “Plug & Play”.

The result of OOA on the "VHE Use Case - Remote Service Development" will be introduced in Table 8 and Table 9. The original requirements will also be traced. Table 8 introduces the actors and Table 9 summarises the analysis. Figure 9 will clarify the results using UML graphics.

In Figure 9, the system, which already presents "Supports Automatic Service" (see Figure 5), now has the four separate extension points, which introduces new APIs as follows:

4. "Supports Home Services" - this API has already been introduced with the GenericUI service concept.

5. "Supports HA Devices", (HA = Home Automation) - an automatic system that supports legacy devices used in home automation, to be added and removed as a homeowner wishes. Appropriate information on the changes in the fittings at home is provided for the contracted service provider.

6. "Supports HA Platforms" - an automatic system supporting a legacy automation platform for home automation, to be added and removed as a homeowner wishes. Appropriate information on the changes in the fittings is provided to the contracted service provider.

7. "Supports Remote Development" - an automatic system that supports the service developer and service provider in having the necessary information and tools to remotely maintain home automation on behalf of a homeowner.

Table 8. Actors of the "Use Case - Remote Service Development".

Tracing the VHE Use Cases	*Actor*	*Subclass*	*Super class*	*Attributes*
NA	Home Owner	NA	Human	Primary
"the remote service developer" "a developer can develop a suitable service application in off-line state"	Service Developer	NA	Human	Primary
"The service provider offering remote service development"	Service Provider	NA	Human	Primary
"Automation platform with several automation devices"	Automation Platform	VHE_Persistent	Daemon	Secondary
"Home server (also serving as gateway)"	Home Server	VHE_Resident	Daemon	Secondary

Table 9. Functional decomposition and APIs of the "Use Case - Remote Service Development".

Tracing the VHE Use Cases	*Use Case*	*Actor*	*Included Use Cases*	*API*
"gets a connection to the home server" "receives an information bundle" "developing the application" "the new application is delivered to the home server"	Develops Remotely	Service Developer	Retrieves Information	"Supports Remote Development"
			Interpreters Information	
			Edits Information	
			Stores Information	
"collecting information about current applications" "gathering information from devices in automation platform"	Provides Remote Deve-lopment Service	Home Server	Collects Information	
"information can be completed with an information from an external source"			Completes Information	
"to a suitable format e.g. utilising XML"			Interpreters Information	
"off-line development"			Stores Information	
"a remote service developer is informed that changes has happened in automation hardware"			Informs Changes	
The table will continue				

Continuing Table 9				
Tracing the VHE Use Cases	*Use Case*	*Actor*	*Included Use Cases*	*API*
"when a device is added to the control area of a specific home server"	Consumes Remote Service Development Service	Home Owner	Adds /Removes HA Devices	"Supports HA Devices"
NA			Adds /Removes HA Platforms	"Supports HA Platforms"
NA	Provides Remote Service Development Service	Home Server	Supports Adding /Removing HA Platforms	
NA	Consumes Home Service	Auto-mation Platform	Supports Adding /Removing HA Devices	"Supports HA Devices"
NA			Supports Adding /Removing Home Services	"Supports Home Services"
NA	Provides Home Service	Service Provider	Installs /Uninstalls Home Service	

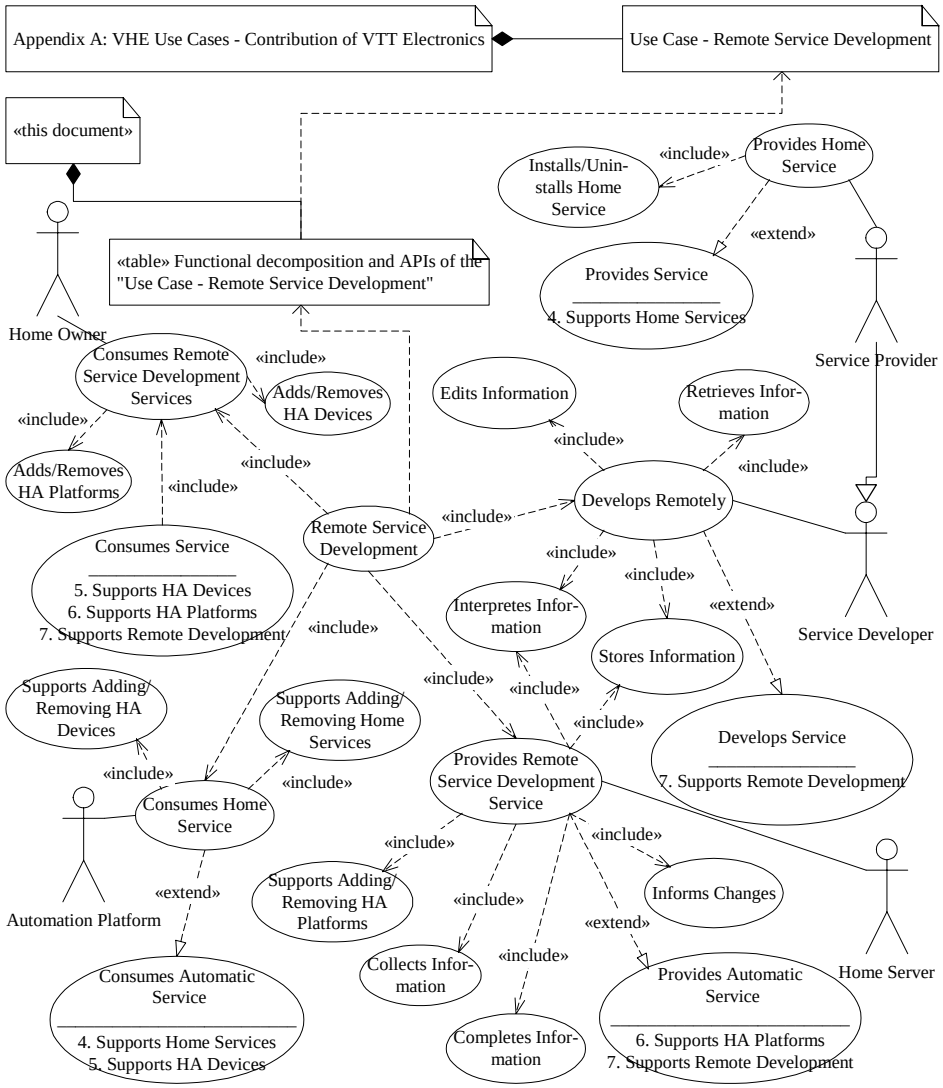

Figure 9. Functional decomposition and APIs of the "Use Case - Remote Service Development".

3.3.3 Elaboration of the "Use Case - Serverless Service Provisioning"

The concept of Serverless Service Provisioning covers the issue of how a set of home appliances, brought together and switched on/off occasionally, can build

up their services in an ad hoc way, and how these services can be made selectable and usable for a user. Additional background to this scenario is presented in [24].

The results of OOA on the "VHE Use Case - Serverless Service Provisioning" will be introduced in Table 10, Table 11 and Figure 10. Table 10 introduces the actors and Table 11 summarises the analysis. Figure 10 clarifies the results using UML graphics.

In Figure 10, the system, which already presents "Supports Automatic Service" (see Figure 5), now has the three extension points, which introduces the APIs:

2. "Supports Service Usage" - this API has already been introduced with the GenericUI service concept.

4. "Supports Home Services" - this API has already been introduced with the GenericUI and Remote Service Development service concept.

8. "Supports Service Discovery" - an appliance supports a directory service system, which is automatically configured amongst the running appliances under the same network domain.

Table 10. Actors of the "Use Case - Serverless Service Provisioning".

Tracing the VHE Use Cases	*Actor*	*Subclass*	*Super class*	*Attributes*
“The user of the system”	User	NA	Human	Primary
“Client (handheld terminal which does not provide services)”	Service Browser	VHE_Service_ Browser	Daemon	Secondary
“Appliance (may be more than one)”	Appliance	VHE_Appliance	Daemon	Secondary

Table 11. Functional decomposition and APIs of the "Use Case - Serverless Service Provisioning".

Tracing the VHE Use Cases	*Use Case*	*Actor*	*Included Use Cases*	*API*
NA	Provides Serverless Service Provisioning Service	Appliance	Initiates/Restores Home Service	"Supports Home Services"
"Service discovery"			Discovers Services	"Supports Service Discovery"
"Initiation of service directory"			Initiates/Restores Directory Service	
"Service discovery of the client"	Consumes Serverless Service Provisioning Service	Service Browser	Discovers Services	
"request service list" "retrieval of the service list"			Retrieves Service Items	"Supports Service Usage"
"the client displays the service list"			Presents Service Items	
"activates one of the service"	Service Usage	User	Selects Service Item	
NA			Uses Service	

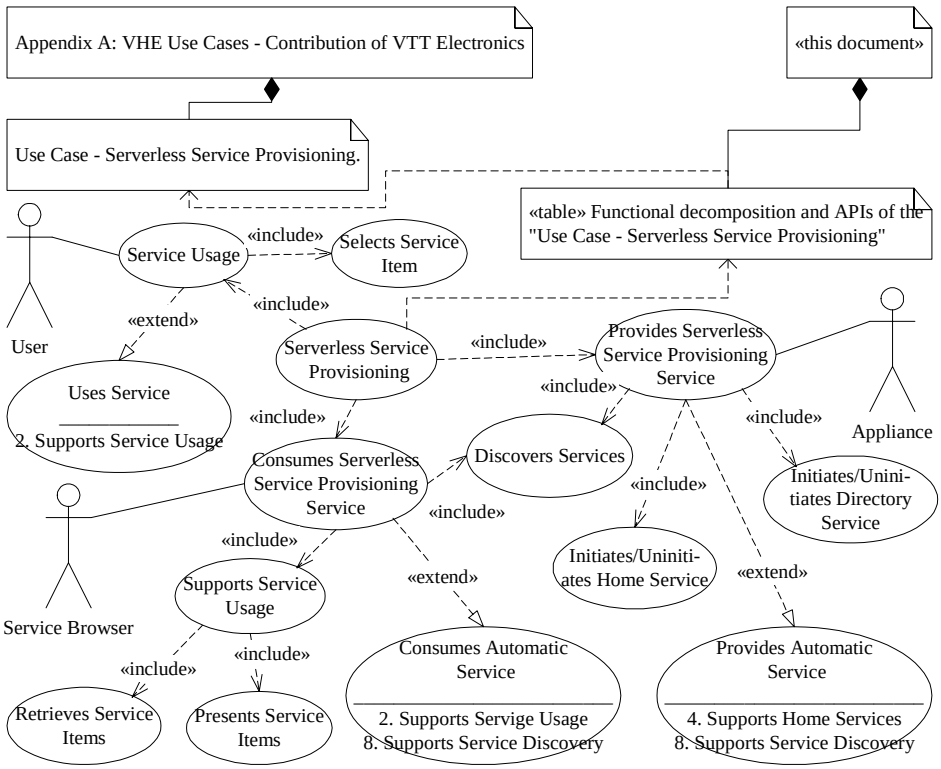

Figure 10. Functional decomposition and APIs of the "Use Case - Serverless Service Provisioning".

3.4 Summary of Conceptual Architecture Design

Conceptual architecture summarises the conceptual design and will advance the deployment of the architecture, i.e. in *concretising* the architecture. Now, the conceptual architecture of VHE Middleware will be drawn up. The result is presented in Figure 11. In the figure, the subsystem named the "VHE Middleware for Core Services" provides VHE Middleware compatibility with the existing and legacy systems. This will support the idea of the “Core Middleware” presented in Section 1.1.4 and, correspondingly, is stereotyped as «core middleware». A realisation of the "VHE Middleware for Core Services" is named the "VHE Core Service". The «core middleware» can be extended. The extensions will inherit all the models presented with the «core middleware» and

will be stereotyped as «upper middleware», respectively. This supports the idea of the “Upper Middleware” introduced in Section 1.1.4.

A set of non-functional requirements is also set for the VHE Middleware. These requirements have been introduced in Section 3.2. These requirements, as well as the functional requirements, will be derived by the extended subsystems - the requirements are set for the whole VHE Middleware. However, both sets of requirements may become overridden, nullified, modified or added to by the extended system, as the principles of inheritance and polymorphism of the OO methodology claims.

Following the UML’s subsystem specification [12], every modelling package of the conceptual architecture contains modelling elements in the three compartments (see Figure 11): “Specification Elements”, “Realisation Elements”, and “Interface Elements”. In this case, the “Specification Elements” compartment includes functional and non-functional requirements elements. The “Realisation Elements” compartment includes “Architecture Model” (AM), “Static Model” (SM), and “Dynamic Model” (DM), as the OO modelling process instructs. Finally, the “Interface Elements” compartment (not explicitly named in Figure 11) includes the interface definitions of the VHE system - the VHE system’s APIs. A modelling element, inside a compartment or independent subsystem, may include or directly refer to its corresponding design document or realising system.

The subsystem packages of the conceptual architecture in Figure 11 will be introduced next.

3.4.1 VHE Middleware for Core Services

The existing or legacy system intended to be compatible with the VHE-system must implement the next API:

1. "Supports Automatic Service" - underlying legacy or existing system is able to provide a Client-Server system that can be extended by the different VHE Middleware extensions.

3.4.2 VHE Middleware for GenericUI

The subsystem "VHE Middleware for GenericUI" extends the "VHE Middleware for Core Services" subsystem by supporting the functional requirements set by the "GenericUI" service concept. The support will be provided by the three APIs that were defined as follows:

2. "Supports Service Usage" - a client is able to retrieve a list of available services and to present it to a user. The system also supports user selection of a service.

3. "Supports GenericUI Service" - a system is able to support client devices in the device-specific way of presentation of appliances' UIs, at least to present a view of but also to control and monitor a specific service that the appliance provides.

4. "Supports Home Services" - an appliance that already provides a specific service is also able to provide a description of its public data, to handle control events and to provide monitoring events to allow it to fulfil its responsibilities.

3.4.3 VHE Middleware for Remote Service Development

The subsystem "VHE Middleware for Remote Service Development" extends the "VHE Middleware for Core Services" subsystem by supporting the functioning and requirements set by the "Remote Service Development"-service concept. The support is to be provided by the four APIs that were defined as follows:

4. "Supports Home Services" - this API has already been introduced with the GenericUI service concept.

5. "Supports HA Devices" - an automatic system that supports legacy devices used in home automation, to be added and removed as a homeowner wishes. Appropriate information on the changes in the fittings at home is provided for the contracted service provider.

6. "Supports HA Platforms" - an automatic system supporting a legacy automation platform for home automation, to be added and removed as a homeowner wishes. Appropriate information on the changes in the fittings is provided to the contracted service provider.

7. "Supports Remote Development" - an automatic system that supports the service developer and service provider in having the necessary information and tools to remotely maintain home automation on behalf of a homeowner.

3.4.4 VHE Middleware for Serverless Service Provisioning

The subsystem "VHE Middleware for Serverless Service Provisioning" extends the "VHE Middleware for Core Services" subsystem by supporting the functioning and requirements set by the "Serverless Service Provisioning" service concept. The support is to be provided by the four APIs that were defined as follows:

2. "Supports Service Usage" - this API has already been introduced with the GenericUI concept.

4. "Supports Home Services" - this API has already been introduced with the GenericUI and Remote Service Development service concepts.

8. "Supports Service Discovery" - an appliance supports a directory service system, which is automatically configured amongst the running appliances under the same network domain.

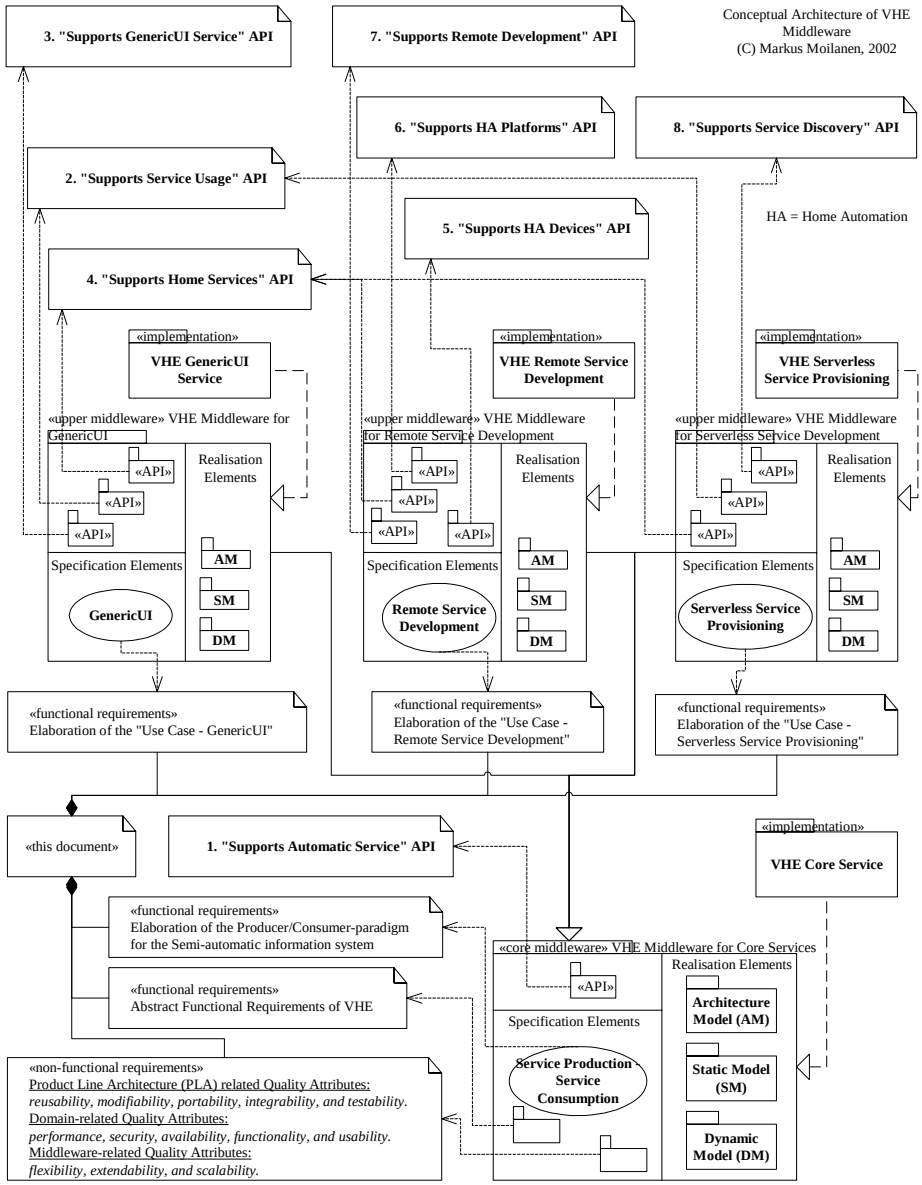

Figure 11. Architecture concept of the VHE Middleware.

3.5 Chapter Summary

This chapter has presented an OOA on a set of the user's scenarios called the VHE Use Cases. These scenarios describe the functioning that is expected for

fulfilling the idea of the VHE presented by the VHE Middleware project. The presented analysis only dealt with three VHE Use Cases from the total of twelve authored by the VHE Middleware project. However, it can be assumed that a great deal of common functionality was found, and that the findings will probably cover most of the functionality of the other nine VHE Use Cases.

The proposed method developed in Chapter 2 was deployed in the analysis and thus **Step 2** introduced in Chapter 1.3 was initialised. The method helped in layering and defining the APIs of the VHE system. The result was an architecture concept of the VHE Middleware. The concept was expressed as a UML subsystems diagram. The results included the definitions of the VHE Middleware conceptual subsystems, their hierarchy, and the appropriate API definitions for deploying the concept.

Further development of the VHE Middleware's architecture will comprise 1) the further development of the presented APIs, and 2) the prototype implementation to verify the APIs. The APIs' development will consider the evaluation of the elements for the design model, and the definitions of the functioning that the elements should provide. These will be presented in the next chapters.

4. APIs of VHE Middleware

In the previous chapter, the VHE Middleware APIs were defined for supporting the compatibility of the different implementations of the VHE systems. Eight of the VHE Middleware's APIs were defined. This chapter will finalise **Step 2** initialised in the previous chapter. The development of the next four APIs will be presented:

1. "Supports Automatic Service" - underlying legacy system is able to provide the defined Client-Server system to be extended by the different VHE Middleware's extensions.

2. "Supports Service Usage" - a client is able to retrieve a list of available services and to present it to a user. The system also supports user selection of a service.

3. "Supports GenericUI Service" - a system is able to support client devices in the device-specific way of presentation of appliances' UIs, at least to present a view of but also to control and monitor a specific service that the appliance provides.

4. "Supports Home Services" - an appliance that already provides a specific service is also able to provide a description of its public data, to handle control events and to provide monitoring events to allow it to fulfil its responsibilities.

The very powerful methodology of the OOA will be deployed again. During the OOA, the systems objects will be found and the functioning reflecting the functional requirement of the system will be illustrated. In this way, the system’s APIs will be modelled. API definitions like component interfaces and sequence diagrams will be expressed in using OCL. OCL has been part of UML since its beginning. OCL lets express conditions on an invocation in a formally defined way. Invariants, preconditions, postconditions, whether an object reference is allowed to be null, and some other restrictions can be defined in using OCL. OCL adds a necessary level of detail to models. OCL is not a programming language, however, guards, conditions, loops and types can be expressed.

4.1 "Supports Automatic Service" API

It was stated in Chapter 2 that the existing technology represents the status quo when a new system is to be developed - the VHE systems will be built on the existing communication and service distribution paradigms. Many different distribution concepts and platforms exist today and a few promising mobile ones are under development. Thus, it is not reasonable to bind the “Core Middleware” to any existing distribution concept, or even build a new one, but rather to support its compatibility with those that exist and those that will come. Most likely, defining the semantics to be used when discussing those platforms and their services is definitely the best way to provide the compatibility between the underlying heterogeneous systems. Because the extended systems will override most of the operations presented by this API, the operations should not be explicitly specified. For instance, when an OSGi-based [6] solution is used to provide the necessary core functioning, its definitions of the operations are sufficient - even if some of the function groups presented in our model will not be implemented by the OSGi. However, the missing function groups might be defined in future versions of OSGi - e.g. the definition of standardised support for smart card -based security. Some work concerning the smart card technique and the security issue is already being carried out by the VHE Middleware project [25, 26]. The communication can be built on any messaging system - e.g. JMS (Java Messaging Service) - or on pure TCP or UDP of IP protocol stack. Because the underlying Client/Server system will remain platform specific, the *Message Based architecture style* [15] is selected at this phase of system modelling. When a VHE service concept is actually implemented, the other style - e.g. *Call-Return architecture styles* [15] - could be chosen to support RMI-based realisations (Remote Methods Invocation).

In the solution suggested now, the heterogeneous distributed Middleware systems that may be used to implement this API will be supported by presenting the system's states. The state support makes it possible to extend the system to control and monitor the underlying system. The extended systems may present

more states as necessary. A peek at Table 14 (on Page 70) and Figure 13 (on Page 72) may help in understanding the concept of the states.

The results of the OOA will be presented as follows: Table 12 introduces the actors, Table 13 introduces the collaborating objects, and Table 14 summarises the suggested semantics API of "Supports Automatic Service", its operations and corresponding interfaces.

Table 12. Actors of the Automatic Service.

Actor	*Subclass*	*Super class*	*Attributes*	*Description*
User	NA	Human	Primary	A user of the system
Server	VHE_ Server	Daemon	Secondary	A system daemon running on Server Device, e.g. Set-Top Box, PDA, PC, etc.
Client	VHE_ Client	Daemon	Secondary	A system daemon running on Client Device, e.g. PDA, PC, etc.
Service Developer	NA	Human	Primary	A human role to develop services

One way of arranging the Client-Server-architecture-based solution is introduced by the author [27]. The author has also suggested a set of management functions to be supported with Client-Server architecture. The management functions can be extended to cover the issues of *resource management*, *security management*, *distribution management* (client software delivering service), *advertising management* (registering/unregistering services), and *profile management*. Moreover, according to the author, these management functions should all be maintained according to the appropriate management policies. A demo program, which introduces the approach, was provided. The solution also provides a concept and realisation of the distribution transparency to be used to harmonise

the communication over a heterogeneous networked and mobile environment. Unfortunately, the states modelled in this architecture do not exactly fit the sates that will be introduced in the solution presented in this publication - correcting the inconsistency should be taken care of during the further development of both solutions.

In Table 13, a column named “Object” names the instance of class giving some kind of friendly name for the discussion. Furthermore, the OO concept of “Multiple Inheritance” will exist between the columns of “Subclass” and “Super class”. This concept is illustrated in Figure 12. The class “VHE_Class_3” inherits both the classes “VHE_Class_1” and “VHE_Class_2”. Realising the concept of multiple inheritance is directly supported by C and C++-programming languages and indirectly supported by the “interface implementation” concept of Java. The latter is congruent with the concept of the “Interface” modelling pattern defined by UML - implementation of the concept of multiple inheritance is not a problem at all.

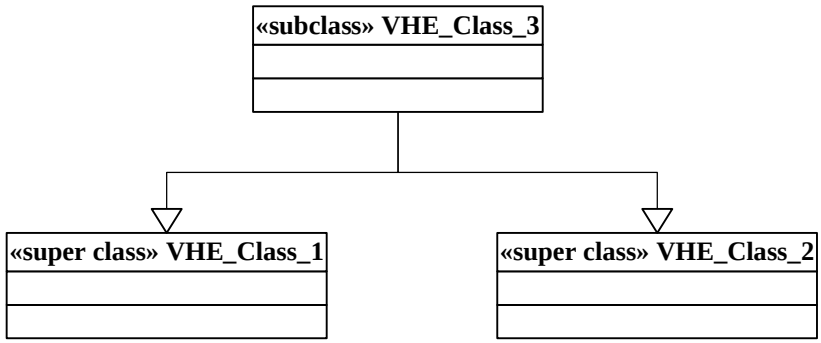

Figure 12. Concept of “Multiple Inheritance” of the OO methodology used to express inheritance hierarchies of classes in Tables and texts through this book.

Table 13. Objects of the Automatic Service.

Description	*Object*	*Subclass*	*Super class*
A device which is VHE-compatible	Device	VHE_Device	Any device
A server device offering a service	Server device	VHE_Server	VHE_Device
A client device capable of retrieving and presenting a service	Client device	VHE_Client	VHE_Device
Connection over network	Communi-cation channel	VHE_Channel	Any connection
An object which can be transferred through a network (like a Java "Serializable" object)	Streamable object	VHE_ Streamable	Any object
An item object	Item	VHE_Item	VHE_String
An object which contains information on a service	Service item	VHE_Sevice_ Item	VHE_Item
A string object which is a streamable type	String object	VHE_String	Any string VHE_Streamable
A plain event-type	Event	VHE_Event	Any event VHE_Streamable
Unique identifier. May contain a name part and GUID part.	uid	VHE_UID	VHE_String
Unique identifier of a service subscriber	ss_uid	NA	VHE_UID
Unique identifier of a service user	su_uid	NA	VHE_UID
Unique identifier of a service provider	sp_uid	NA	VHE_UID

Table 14. Operations of the Automatic Service.

Actor	*Use Case*	*Description of operations*	*Inter-face*
Service User	Starts Service	Service User uses a VHE service that is made available by the Service Provider. The users are instructed on how to use the system	Instructions for VHE Service Users
	Uses Service		
	Closes Service		
Will be overridden by the imple-mentator of a VHE device - Client or Server typed actor	Supports On/ Off	Platform-specific operations concerning the means of switching a device on/off in a managed way. When a VHE device is switched on, it must be initialised. When the device is switched off, it must be allowed to reach the safe state to be shut down. The system's states "ON" and "OFF" should be maintained	VHE_Device
	Configures Service	Platform-specific operations to configure a device's parameters - e.g. the identity of the device and its installer. Smart card functions might be supported. The system's states "UNCONFIGURED", "CONFIGURING", "CONFIGURED" and "UNCONFIGURING" should be maintained by a VHE device	
	Initiates/ Uninitiates Communi-cation	Platform-specific operations to initiate/uninitiates the communication channels of the device. The system's states "DISCONNECTED", "CONNECTING", "CONNECTED" and "DISCONNECTING" should be maintained by a VHE device	

The table will continue

Continuing Table 14			
Actor	*Use Case*	*Description of operations*	*Interface*
Service Developer	Develops Service	Service Developer develops services by developing the corresponding applications according the given instructions and tools	Instructions for Service Developers
Server	Initiates/ Uninitiates Service	Platform-specific operations to initiate/uninitiate a service. The system's states "STOPPED", "STARTING", "STARTED" and "STOPPING" should be maintained	VHE_Server extends VHE_Device
	Registers/ Unregisters Service	Platform specific operations to register/unregister services. This means a way of notifying the availability of a new service to the associated participants. When the service is restored, it must be unregistered correspondingly. The system's states "UNREGISTERED", "REGISTERING", "REGISTERED", and "UNREGISTERING" should be maintained correspondingly	
Client	Retrieves Service	Platform-specific operations to retrieve client software and enable the use of an offered service. The subscriber's identity is notified during the retrieving. The system's states "UNSERVED", "RETRIEVING", and "SERVED" and "REMOVING" should be maintained	VHE_Client extends VHE_Device
	Presents Service	Platform-specific operations to install the client-side software. The system's states "UNINSTALLED", "INSTALLING", "INSTALLED", and "UNINSTALLING" should be maintained	

4.1.1 Interfaces of the "Supports Automatic Service" API

Interfaces of “Supports Automatic Service” and their hierarchy are illustrated in Figure 13. The figure shows that supported system states are defined as public constants (marked with +). All operations have been left platform specific - the extended system may present them as the most appropriate for the extensions.

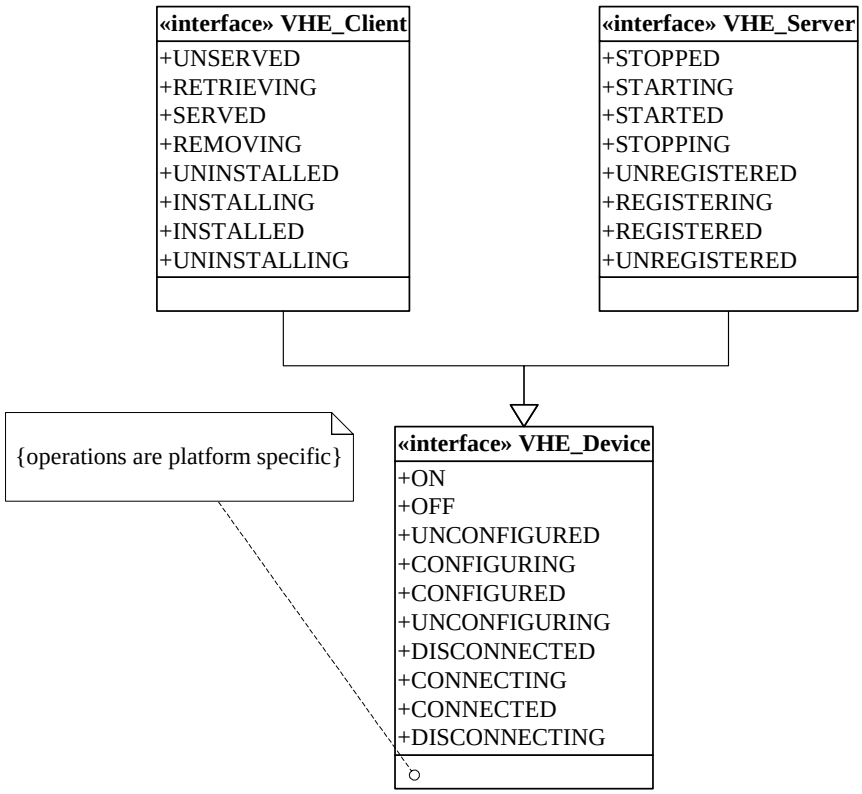

Figure 13. Interfaces and their inheritance hierarchy of the “Supports Automatic Service” API. The public constants express the states of the VHE-system.

4.2 "Supports Service Usage" API

The "Supports Service Usage" API provides a client with the ability to retrieve a list of available services and present it to a user. The system also supports the user selection of a service.

The results of the OOA will be presented as follows: Table 15 introduces the actors, Table 16 introduces new objects that are necessary in presenting the services which this API offers, and Table 17 describes the operations.

Table 15. Actors of the Service Usage.

Tracing the VHE Use Cases	*Actor*	*Subclass*	*Super class*	*Attributes*
"User of the system"	User	Service User	Human	Primary
"Handheld device"	Service Browser	VHE_Service_ Browser	Daemon	Secondary

Table 16. New objects for the Service Usage.

Description	*Object*	*Subclass*	*Super class*
A client device is capable of retrieving and presenting a service list	Service Browser	VHE_Service_ Browser	VHE_Client
An item on the service list. Contains all the necessary information for reaching and deploying a particular service	service_item	NA	VHE_Service _Item
Identity of a service provider	broker_uid	NA	VHE_UID
Identity of a service consumer	browser_uid	NA	VHE_UID
A list containing service items	service_list	NA	VHE_Service _List

Table 17. Responsibilities and interfaces of the Service Usage.

Actor	*Use Case*	*Description of operations*	*Inter-face*
Service Browser	Retrieves Service Items	When the state "CONNECTED" of the underlying VHE_Client occurs, indicating the system's capability to communicate, Service Browser will retrieve a service list: **service_list:= retrieveServiceList(browser_uid: VHE_UID, broker_uid: VHE_UID): VHE_Service_List** The operation returns a service list object, or null if the operation fails. If the operation was successful, meaning that a service list object was returned, the system moves on to the state "LISTING", otherwise the "null list object returned exception" will appear	VHE_Service_Browser extends VHE_Client
	Presents Service Items	If the browser's state "LISTING" is perceived, Service Browser will start presenting the service list to a user and will wait until the user has made a selection on the presented list: **service_item:= presentServiceItems (_service_list: VHE_Service_List): VHE_Service_Item** When the user has selected a service item from the service list, the browser moves on in its state "RETRIEVING", or, in case of failure, the "service item cannot be selected exception" will appear	
Service User	Selects Service Item	When a user perceives a list of service items appearing (during the browser's state "LISTING"), selection of a service item will be made by Service User	Instructions for VHE Service Users

4.2.1 Interfaces of the "Supports Service Usage" API

The component that implements the interface VHE_Service_Browser is responsible for providing the Service Usage operations on a client platform. Figure 14 illustrates this interface. The figure shows how the VHE_Service_Browser extends the VHE_Client interface. By this, it will receive the support for the states of the underlying system.

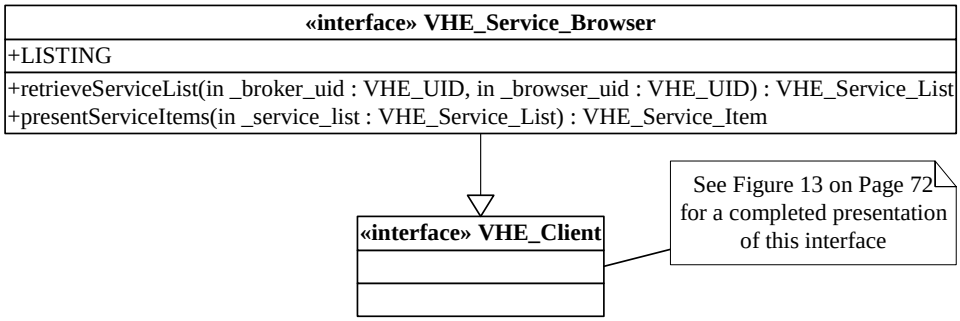

Figure 14. Interface VHE_Service_Browser extends the interface VHE_Client.

4.3 "Supports GenericUI Service" API

"Supports GenericUI Service" - a system is able to support client devices in the device-specific way of presentation of appliances' UIs, at least to present a view of but also to control and monitor a specific service that the appliance provides.

The results of the OOA will be presented as follows: Table 18 introduces the actors, Table 19 introduces the objects and Table 20 summarises the suggested APIs.

Table 18. Actors of the GenericUI Service.

Tracing the VHE Use Cases	*Actor*	*Subclass*	*Super class*	*Attributes*
"a home server also serving as a gateway and providing the GenericUI-service"	UI Broker	VHE_UI _Broker	Daemon	Secondary
"Handheld device"	Service Browser	VHE_Service_ Browser	Daemon	Secondary

Table 19. New objects for the GenericUI Service.

Description	*Object*	*Subclass*	*Super class*
A server device is acting as the UI Broker between appliances and client devices	UI Broker	VHE_UI_Broker	VHE_Server
The event which is received from the UI Broker	push_event	VHE_Push_Event	VHE_Event
The event which tells the service browsers to update the presented UI	"UPDATE_UI"	NA	push_event

Table 20. Operations of the GenericUI Service.

Actor	*Use Case*	*Description of operations*
UI Broker	Maintains Transcodecs	Will be defined in the future versions
	Maintains Profiles	Will be defined in the future versions
	Maintains UI Descriptions: - Retrieves and Updates UI Description - Stores UI Description	For retrieving a UI description for a specific appliance, the UI Broker can use the next operation: **_UI_description:= retrieveUIDescription(_app_uid: VHE_UID, _description_id: XML_String): VHE_UI_Description** The _app_uid is the identity of an appliance and the _description_id is the identity of the specific UI description presented by or suitable for the appliance. If there is more than one option, the parameter _description_id can be for selecting one. The null value of the _description_id will retrieve the default UI description. This operation doesn't define how The UI Broker stores the UI description for later retrieval: **_ui_stored:= storeUIDescription(_app_uid: VHE_UID, _ui_description: VHE_UI_Description, _description_id: VHE_String): Boolean**
	Handles Monitoring Events	The underlying Server passes all events that are identified as monitoring events to this operation. The operation returns TRUE if the event was handled. **monitoring_event_handled:= handleMonitoringEvent(_appl_uid: VHE_UID, _description_id: XML_String, _data_description: VHE_Data_Description, _monitoring_event: VHE_Monitoring_Event): Boolean**
The table will continue		

Continuing Table 20		
Actor	*Use Case*	*Description of operations*
UI Broker	Handles Monitoring Events	Only the monitoring event "DATA_CHANGED" must be handled at the time. Because some data in some appliance has been changed, the UI description for the appliance in question must be updated. After these operations, the UI Broker will inform its clients by generating the push event of "UPDATE_UI"
	Passes Events	All coming messages that can be interpreted by the underlying Server to be legal in the system are directed to this operation. If a message is interpreted as containing events, the event type will be identified. If the event is a monitoring event, the handleMonitoringEvents() -operation will be called. If the event is a control event, it will be directed to the appliance whose uid was identified in the message. The UI Broker can pass the events as: **passEvent(_from_uid: VHE_UID, to_uid: VHE_UID, _event: VHE_Event): Boolean**
Service Browser	Fetches and Updates UI	The UI Broker can transcode an UI description to the specific UI format by the next operation: **_ui:= transcodeUIDescription(_ui_description: VHE_UI_Description, _data_description[]: VHE_Data_Description, _format_id: XML_String): VHE_UI** The parameter _ui_description is the UI Description to be transcoded. The parameter _data_description[] is the list of VHE_Data_Descriptions to be included in the _ui_description
	Presents UI	Platform-specific operation
	Handles Push Events	Will be defined in the future versions
	Generates Control Events	Platform-specific operation

4.3.1 Interfaces of the "Supports GenericUI Service" API

The description of the operations of the GenericUI service is drawn as an interface definition of “VHE_UI_Broker” in Figure 15. The parameters in the operations handleMonitoringEvent(), storeUIDescription() and transcodeUIDescription() are suppressed because of their required space.

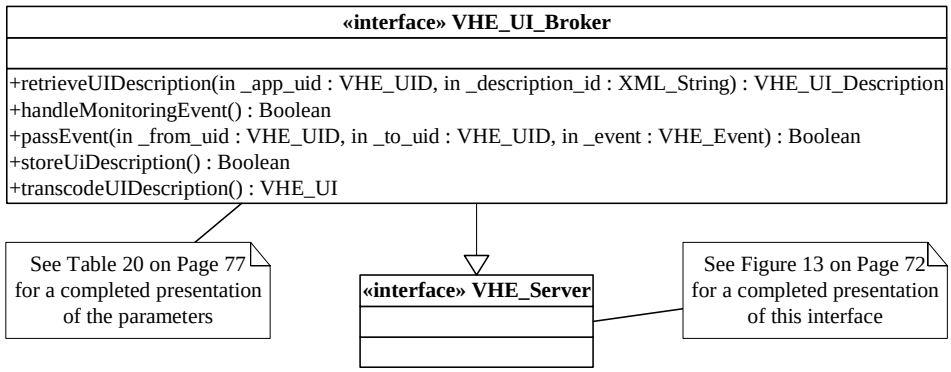

Figure 15. Interface VHE_UI_Broker extends the interface VHE_Server.

4.4 "Supports Home Service" API

The "Supports Home Services" API supports the appliance that provides a service offering a description of its public data, to handle control events and to provide monitoring events to allow it to fulfil its responsibilities. The results of the OOA will be presented as follows: Table 21 introduces the actors, Table 22 introduces the objects, and Table 23 summarises the suggested API.

Table 21. Actors of the Home Service.

Tracing the VHE Use Cases	*Actor*	*Subclass*	*Super class*	*Attributes*
“Appliance more than one”	App-liance	VHE_ Appliance	Daemon	Secondary
“VHE service subscriber”	Home Owner	NA	Human	Primary

Table 22. Objects of the Home Service.

Description	*Object*	*Subclass*	*Super class*
A server device on which a home service runs	Appliance	VHE_Appliance	VHE_Server
Identity of a broker	broker_uid	NA	VHE_UID
Identity of an appliance	appl_uid	NA	VHE_UID
An event which an object sends to a collaborator for informing changes to its data	monitoring_ event	VHE_Monito-ring_Event	VHE_Event
The event which is received from a service browser	control_ event	VHE_Control _Event	VHE_Event
A string in XML format, and it is a streamable type	XML	XML_String	Org.W3C.co m /document/ VHE_ Streamable
An XML string which describes the identity of a data item	description_id	NA	XML_String
An XML string which describes the type of a data item	data_type	NA	XML_String
An XML string which describes the value of a data item	data_value	NA	XML_String
An XML string which describes the access type a data item, e.g. read, write	data_access	NA	XML_String
The table will continue			

Continuing Table 22			
Description	*Object*	*Interface*	*Super Class*
A string an appliance maintains for presenting its public data in XML format. It contains four parts: description_id, data_type, data_value, and data_access	data_description	VHE_Data _Description	Description_id, data_type, data_value, data_access
A generic UI presented in XML format. Ready to transcode to UI	UI Description	VHE_UI _Description	XML_String
A description is presented in XML 1.0 format	“XML_10”	NA	XML_String
A description is presented in HTML 1.1 format	“HTML_11”	NA	XML_String
A description is presented in WML 1.0 format	“WML_10”	NA	XML_String
A description is presented in UIML 1.0 format	“UIML_10”	NA	XML_String
A description is presented in Java2 byte code (a Jini proxy)	“JAVA_2”	NA	XML_String
Instance of monitoring event	"DATA_ CHANGED"	NA	monitoring_ event
Instance of control event	"UPDATE_ DATA"	NA	control_event

Table 23. Operations of the Home Service.

Actor	*Use Case*	*Description of operations*	*Inter-face*
Appliance	Generates Monitoring Events	A monitoring event is sent to the appropriate UI Broker expressed by the parameter _broker_uid. Only the "DATA_CHANGED" event is defined by now. The identity of the appliance itself is expressed by _appl_uid. The _description_id and the _data_description express the event-related data. The operation returns TRUE if the sending was successful: **_event_sent_:= sendMonitoringEvent(_monitoring_event: VHE_Monitoring_Event, _appl_uid: VHE_UID, _broker_uid: VHE_UID, _description_id: XML_String, _data_description: VHE_Data_Description): Boolean** A private operation of **_data_description:= getDataDescription(_description_id: VHE_String): VHE_Data_Description** can retrieve a data description object from the appliance by a description id. The parameter _description_id selects one of a set of named descriptions. A value of “null” will return the default (or start) description. The operation can return a null value if the operation cannot be served. The behaviour is illustrated in Figure 16	VHE_Appliance extends VHE_Server (see Figure 18)
The table will continue			

Continuing Table 23			
Actor	*Use Case*	*Description of operations*	*Inter-face*
Appliance	Handles Control Events	All arrived messages which can be interpreted by the Server to be legal in the system are directed to the handleControlEvent() operation. The operation returns TRUE if the event was handled. **_event_handled := handleControlEvent (_broker_uid: VHE_UID, _description_id: XML_String, _data_description: VHE_Data_Description, _control_event: VHE_Control_Event) : Boolean** If the _control_event is "UPDATE_DATA", the appliance must update the appropriate data description expressed by the _description_id. This is how the appliance can be controlled. On the real change in the appliance's data, the DATA_CHANGED-event will be raised by the Appliance. If the _control_event is "DELIVER_DATA", an appropriate _description_id –related data description must be sent. This is how the UI Broker can retrieve the Appliance's data explicitly. The behaviour is illustrated in Figure 17	VHE_Appliance extends VHE_Server (see Figure 18)
The table will continue			

Continuing Table 23			
Actor	*Use Case*	*Description of operations*	*Inter-face*
Home Owner	Adds/ Removes Device	Home Owner adds/removes a device at home. VHE_Server registers / unregisters the application with the service provider, who must maintain the lookup service for the appliances. If an appropriate extension for the VHE_Server capabilities is presented (e.g. a lease service presented by Serverless Service Provisioning), the registration will be guarded in case of an unexpected removal of a network connection or a device	Instructions for the Home Owner

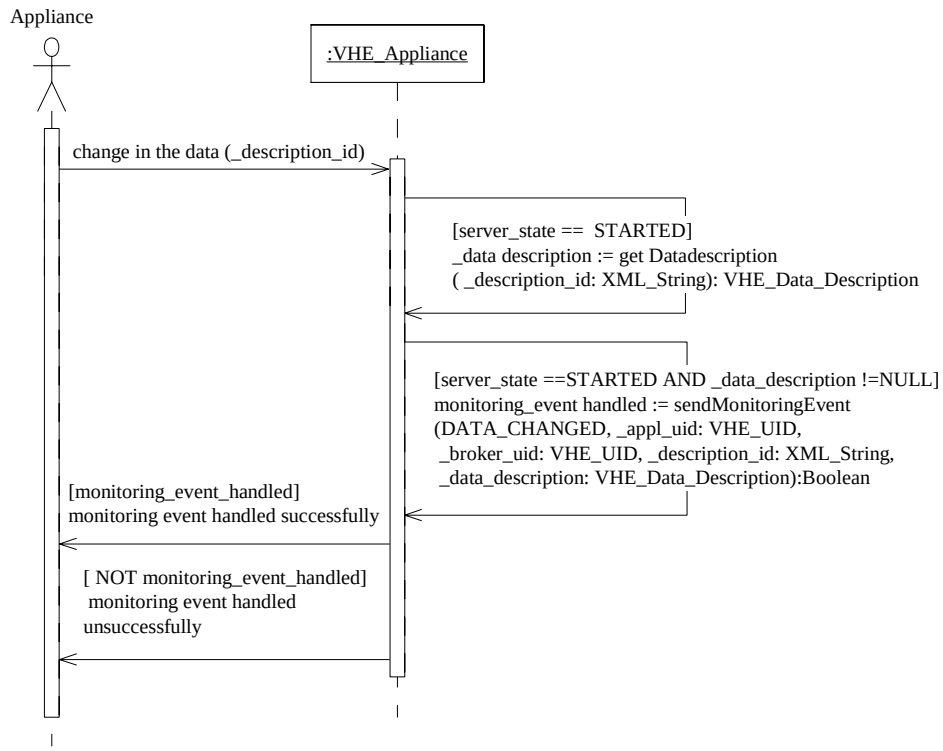

Figure 16. Appliance generates a monitoring event.

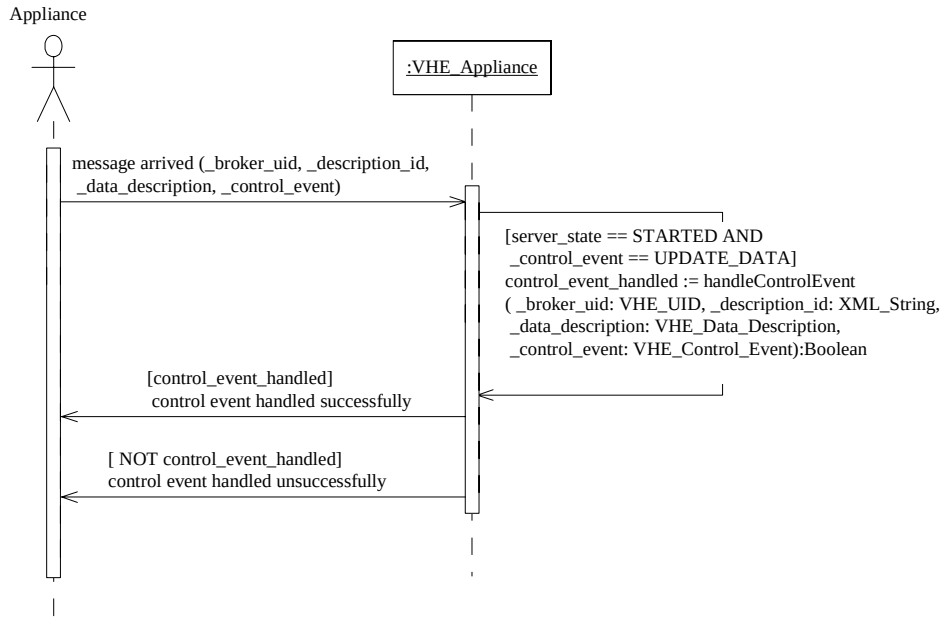

Figure 17. Appliance handles a control event.

4.4.1 Interfaces of the "Supports Home Service" API

The component that implements the interface VHE_Appliance is responsible for providing a home service. The interface is illustrated in Figure 18. The VHE_Appliance interface extends the VHE_Server interface. In this way, the underlying system's states can be notified. Suppressed parameters of handleControlEvent() and sendMonitoringEvent() operations can be seen in Table 23, Figure 16 and Figure 17.

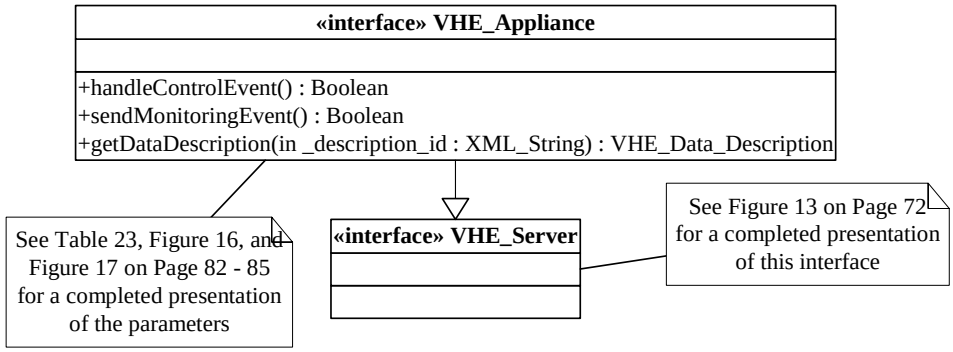

Figure 18. Interface VHE_Appliance extends the interface VHE_Server.

4.5 Chapter Summary

This chapter has presented the API development of the VHE Middleware's architecture concept thus finalising **Step 2** initialised in the previous chapter. The concept was presented in the previous chapter. The result was a set of interface definitions with a set of objects the VHE system needed to provide the functions the conceptual APIs define. The APIs and interfaces developed here are expressed in OCL and, consequently, are platform independent - they can be deployed in any environment in which the VHE-compatible arrangement should take place.

Further development should comprise the building of an implementation of the suggested APIs. With this, the usefulness of the results will be verified, maybe leading to further improvement of the APIs. This will be done by a case study, which we will present next.

5. Case Study

In this chapter, we will conduct a case study of a prototype system. By this, **Step 3** introduced in Chapter 1.3 will be entered. The case is an implementation of the GenericUI service concept and is called the "UIML Service". The study will show how the GenericUI-related APIs should be implemented in UIML-based realisations of the GenericUI service concept. The method to be applied is *Reverse engineering*. According to the Reverse Engineering methodology, by analysing the code of software, its abstractions can be found and the software can be restructured accordingly. Brown et al. [11] have identified several effects of the programming - what they call the "Anti-Patterns". The "Anti-Patterns" are the opposite of "Patterns", and the "Patterns", on their behalf, are the commonly accepted good ways of modelling and programming software. An Anti-pattern called the "Functional Decomposition" can be identified in the prototype software. Typically, this Anti-pattern is caused by the style of prototype programming, where the object-oriented (OO) analysis of the problem and its following OO-architecture design is omitted and replaced with great enthusiasm and creativity for the subject. To resolve this Anti-pattern, the object-oriented decomposition model must be provided; after that, the prototype software's functions can be restructured by the suggested decomposition model. The proposed method will be tested as well as **Step 3** suggests. A full example of prototype software is presented in Appendix B. Unfortunately; the full source code cannot be published here, but some of it can be perceived in the models.

5.1 Decomposition Model of the UIML Service

To provide the object-oriented decomposition model for the UIML Service, the decomposition model of the GenericUI service illustrated in Figure 8 should be developed further. The UIML service concept should understood as an extension of the GenericUI service concept - the UIML service concept will extend the GenericUI service by deploying the UIML language in providing the UI service. The model is illustrated in Figure 19. The figure presents the case-related documentation and the suggested functional decomposition model for the UIML service.

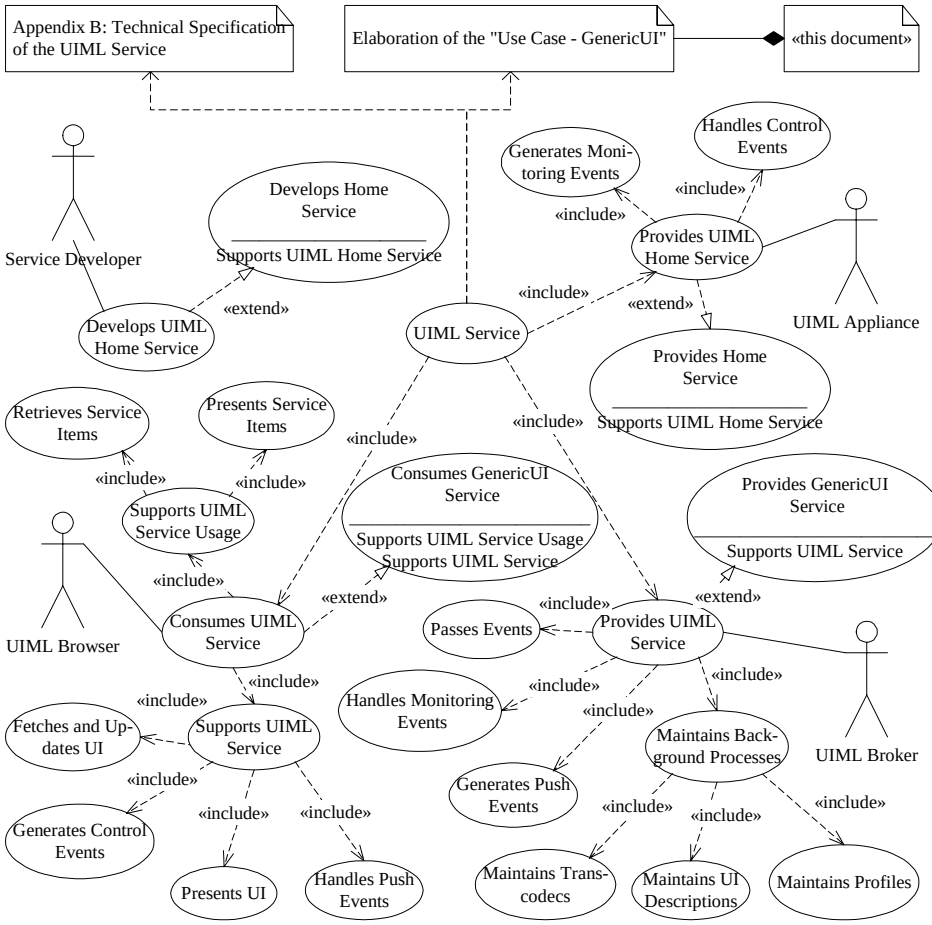

Figure 19. Functional decomposition model of the UIML Service.

5.2 APIs of the UIML Service

The VHE Middleware APIs in Chapter 4 were modelled in using OCL, but the UIML service prototypes is written in Java - the restructured APIs will be Java-specific too. This is not, however, a problem at all because these two “languages” have many similarities, making the APIs easily to read and compare.

5.2.1 "Supports UIML Service Usage" API

In Figure 19, the "Supports UIML Service Usage" API extends the "Supports Service Usage" API in the way that the first one overrides some function groups of the latter, namely the "Retrieves Service Items" and "Presents Service Items". Other function groups will remain platform specific. The results of the restructuring process are as follows: Table 24 introduces the actors, Table 25 introduces the objects, and Table 26 summarises the suggested APIs.

Table 24. Actors of the UIML Service Usage.

Actor	*Subclass*	*Super class*	*Attributes*	*Description*
UIML Browser	VHE_UIML _Browser	VHE_Service_ Browser	Secondary	A daemon running on a client device

Table 25. Objects of the UIML Service Usage.

Description	*Object*	*Subclass*	*Super class*
Main class of application used for UI browsing		JavaUIMLBrowser	java.awt.event.WindowAdapter
The main frame of the JavaUIMLBrowser		Browser-Frame	java.awt.Frame
Guides the reading of UIML document			UIMLReader
Reads peers section of UIML document			UIMLPeer-Reader
Creates Java components from UIML structures		UIMLJava-Renderer	UIMLCallTarget
Any part of UIML structure		UIMLPart	UIMLElement
Base class (root) of complete UIML structure		UIMLPart	UIMLStructure
Adapts user's actions on UI elements to UIMLCalls		UIMLAction Adapter	ItemListener
Base class for all Java UI components		JComponent	java.awt.Container
Class for URL-address			URL
Component that can contain other JComponent-based components			JContainer

Table 26. Operations of the UIML Service Usage.

Actor	*Use Case*	*Description of operations*	*Inter-face*
UIML Browser	Retrieves Service Items	A list of available services will be shown in the UIML Browser (as an UIML-format UI that is dynamically created by the UIML Service) when no service name is specified in the retrieve request. The following interaction is the same for retrieving a specific service, except that in that case the name of the service will be given in the request (a parameter in the URL). BrowserFrame::convertUIMLDescription is called when the user starts the retrieval of a service list or a specific service. Method creates a connection to UIML Service's servlet and, when connection has been established, it reads data through the connection to an InputStream variable. This variable is then converted to objects of classes UIMLReader and UIMLPeerReader for later use in presenting of data. This functioning is illustrated in Figure 20.	UIML_Browser
	Presents Service Items	Presenting of a service list or service UI is initialised when BrowserFrame creates a new UIML JavaRenderer object, giving it UIMLReader and UIML PeerReader as parameters. After this, the method UIML JavaRenderer::renderStructure is called to actually start the presenting. This method calls the renderPart method for each UIMLPart in UIMLStructure received from UIMLReader. renderPart identifies each part and calls CreateXXX method for it (XXX can be, for example, Trigger, TextEntry or List).	
The table will continue			

Continuing Table 26			
Actor	*Use Case*	*Description of operations*	*Inter-face*
UIML Browser	Presents Service Items	In the CreateXXX method, a new JComponent based Java component is created and a new actionListener of type UIML ActionAdapter is added to it with a call to method addActionListener. At this point Java components have been created for every part in UIMLStructure and UI shows the items. Components have actionListeners to be able to receive user selections later. This functioning is illustrated in Figure 21.	UIML_Browser

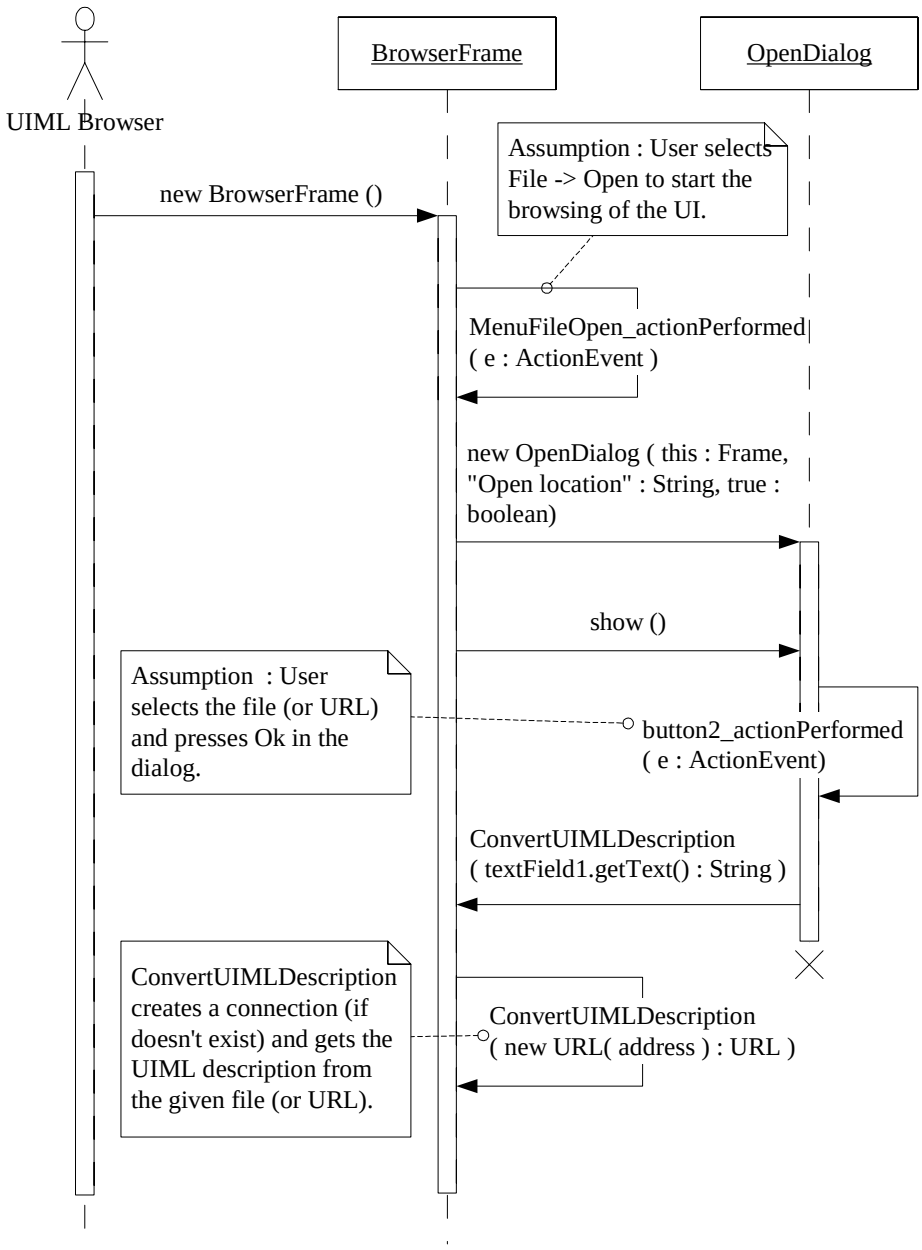

Figure 20. UIML Browser - Retrieves Service Items.

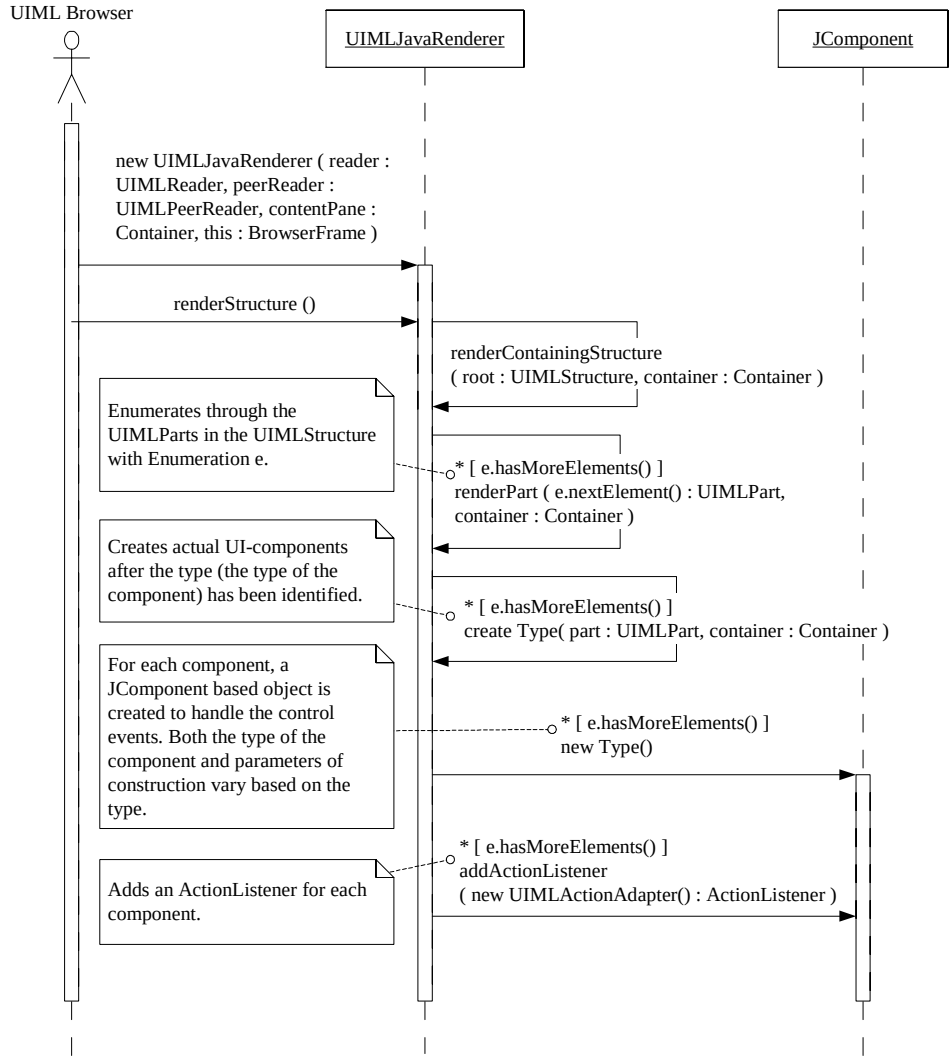

Figure 21. UIML Browser - Presents Service Items.

5.2.2 "Supports UIML Home Service" API

All function groups from the "Supports Home Service" API will be overridden. This is because the UIML presentation of a home service needs the UIML-specific events handling and generation, and, furthermore, a UIML appliance must be able to present its data description in the UIML-supporting format. To

present the OOA, Table 27 introduces the actors, Table 28 introduces the objects, and Table 29 summarises the suggested APIs.

Table 27. Actors of UIML Home Service.

Role	*Sub class*	*Super class*	*Attributes*	*Description*
UIML Appliance	UIML_ Application	VHE_ Appliance	Secondary	A system daemon running on a server device which provides a home service, e.g. TV set's remote control
Service Developer		Human	Primary	A UIML-aware service developer

Table 28. Objects of UIML Home Service.

Description	*Object*	*Subclass*	*Super class*
Application's reply to a method call		UIMLReply	java.io.Serializable
UIML description of application's UI		UIMLDescription	java.io.Serializable
Class for transferring UIML values from and to application		UIMLValues	java.io.Serializable
Interface for all applications that work as a UIML appliance (used in communication with UIML Service)	UIML Appliance	UIML_Application	VHE_Appliance

Table 29. Operations of the UIML Home Service.

Actor	*Use Case*	*Description of operations*	*Inter-face*
UIML Appliance	Generates Monitoring Events	Service registration / unregistration, status monitoring and updates of the UI. Prototype only supports the registration / unregisration events.	UIML_Application
	Handles Control Events	The UIML_Application::invokeMethod is called by the UIML broker and actually starts the control events handling in the application. After processing the event, the application creates a UIMLReply object, which includes application userid, replyvalue and replytype, and returns this object to the UIML broker, which then decides what to do to the UI (for example, to use a cache version, update only values or retrieve a completely new UI from the application). This functioning is illustrated in Figure 22.	
Service Developer	Develops Service	A service developer is aware of the UIML service and has agreed to follow the interface "UIML_Application" in developing the software for devices	Instructions for UIML Service developers

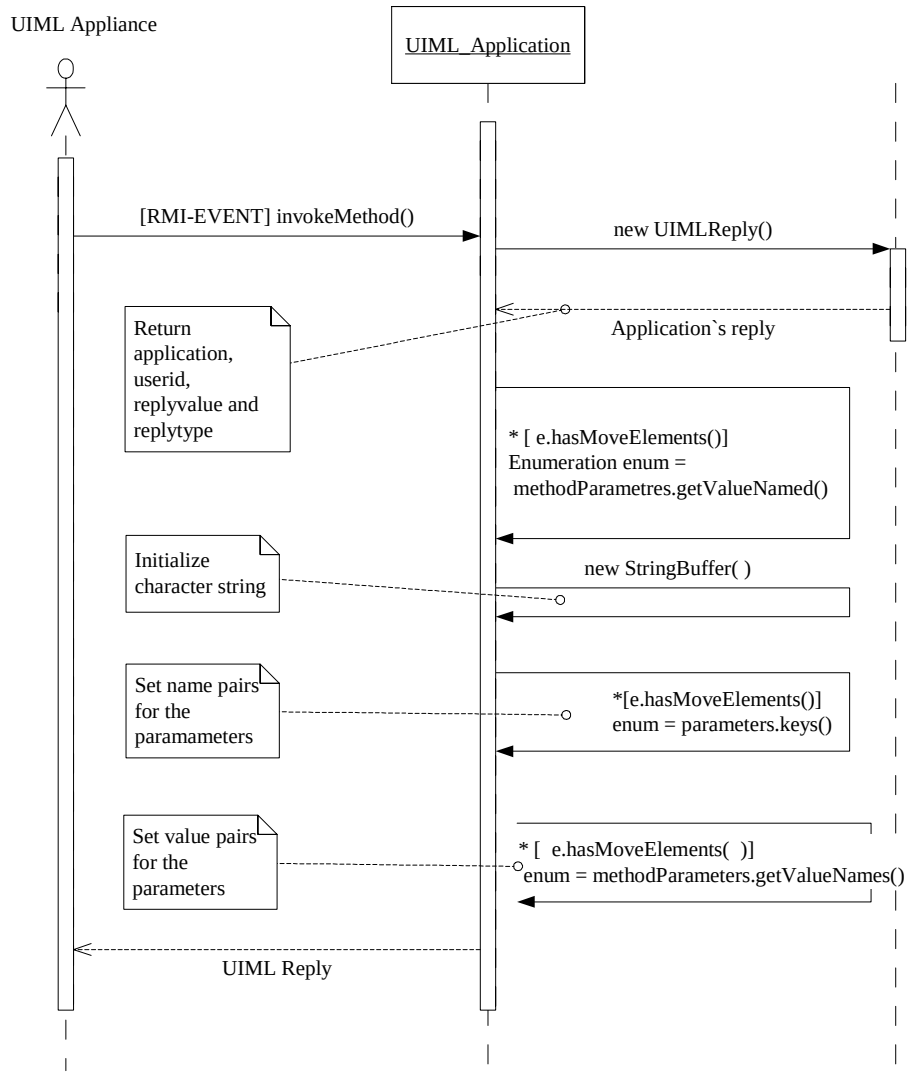

Figure 22. UIML Appliance - Handles Control Events.

5.2.3 "Supports UIML Service" API

The "Supports UIML Service" API relies on the "Supports GenericUI Service" APIs in many operations because of their generality. However, some function groups must work differently and thus will need to be overridden. To present the results of the OOA, Table 30 introduces the actors, Table 31 introduces the objects, and Table 32 summarises the suggested APIs.

Table 30. Actors of the UIML Service.

Actor	*Sub class*	*Super class*	*Attributes*	*Description*
UIML Broker	VHE_UIML _Broker	VHE_UI_Bro-ker	Secondary	A system daemon running on a server device, e.g. Set-Top Box
UIML Browser	VHE_UIML _Browser	VHE_Service_ Browser	Secondary	A system daemon running on a client device, e.g. PDA, PC etc.

Table 31. Objects of the UIML Service.

Description	*Object*	*Subclass*	*Super class*
Main class of component for communication between Java UIMLBrowser and UIMLService	NA	UIMLServlet	javax.servlet. http.HttpServlet
Main class of component UIMLService, implements UIML Broker	NA	UIMLService	UIMLCallTarget
Request from UIMLservlet to UIMLService	NA	UIMLServlet-Request	java.io.Seria-lizable
Response from UIMLService to UIMLServlet	NA	UIMLServletR esponse	java.io.Seria-lizable
The class implements a call to a method as described in the structure section of the UIML document	NA	UIMLCall	UIMLEvalu-atable, UIMLAction

Table 32. Operations of the UIML Service.

Actor	*Use Case*	*Description of operations*	*Inter-face*
UIML Broker	Maintains Transcodecs	Stores and retrieves transcodecs (transcodecs are used for converting UIML to other formats). The prototype only stores transcodecs	UIML_Broker
	Maintains UI Descriptions	The UIML_Application typically reads the data descriptions and parameters on start-up by calling internal methods. These are then saved, for example to member variable String description and Hashtable parameters. The description is returned when the method getDescription is called (by the UIML broker) and the requested parameters when requestValues is called. Both of these methods are part of the UIML Application interface. The UIML broker also stores these descriptions in its private cache for later use. This functioning is illustrated in Figure 23.	
	Maintains UI Profiles	The prototype does not support this function group.	
	Generates Push Events	The prototype does not support this function group.	
	Handles Monitoring Events	The prototype only recognises application registrations and unregistration.	
The table will continue			

Continuing Table 32			
Actor	*Use Case*	*Description of operations*	*Inter-face*
UIML Broker	Passes Events	UIML Browser calls UIMLServlet`s doGet - method. After that, UIMLServlet determines the preferred response format and checks the availability of the UIML broker service. If these items are ok, class UIML ServletRequest constructs an object, demand by HTTP ServletRequest, and sets the preferred response format. After that, UIMLServlet calls the service method from the class UIMLService. This method invokes the service and the event will be passed to the application. Finally, an updated client format description is returned in the response. This interaction is very similar to fetching / updating UI.	UIML Broker
UIML Browser	Fetches and Updates UI	The fetching of a UI description is started when the UIML Browser makes an HTTP request to UIMLServlet (its method doGet is called). UIMLServlet then calls the UIMLService's method service (UIML ServletRequest, UIML ServletResponse). If the request does not include the name of the application or method, a list of available applications is returned. However, if these identifiers do exist, the method invokeMethod is called. This method can result in a need to update the UI description. UIMLService first checks if it has the correct UI description in its cache; if not, it requests it from UIML Application with method UIML Description getDescription(String). This description is returned and stored to cache. This functioning is illustrated in Figure 24.	UIML_Browser
The table will continue			

Continuing Table 32			
Actor	*Use Case*	*Description of operations*	*Inter-face*
	Generates Control Events	When a user presses any button on the UIMLJavaBrowser UI, the method actionPerformed of UIMLActionAdapter is called (check UIML Service Usage - Presents Service Items for details about how the UI and UIMLActionAdapters are created). This method creates a UIMLAction object (of type UIMLCall) based on the UIMLRule object that has been set to UIMLActionAdapter during the creation of the adapter. These rules originate from the UIML description of the appliance and include the valid control events. The control event is passed on by calling the evaluate() method of UIMLCall. This functioning is illustrated in Figure 25.	UIML_Browser
	Presents UI	The description of these operations can be found from UIML Browser - Presents Service Items, and Figure 21.	
	Handles Push Events	The prototype does not support this function group.	

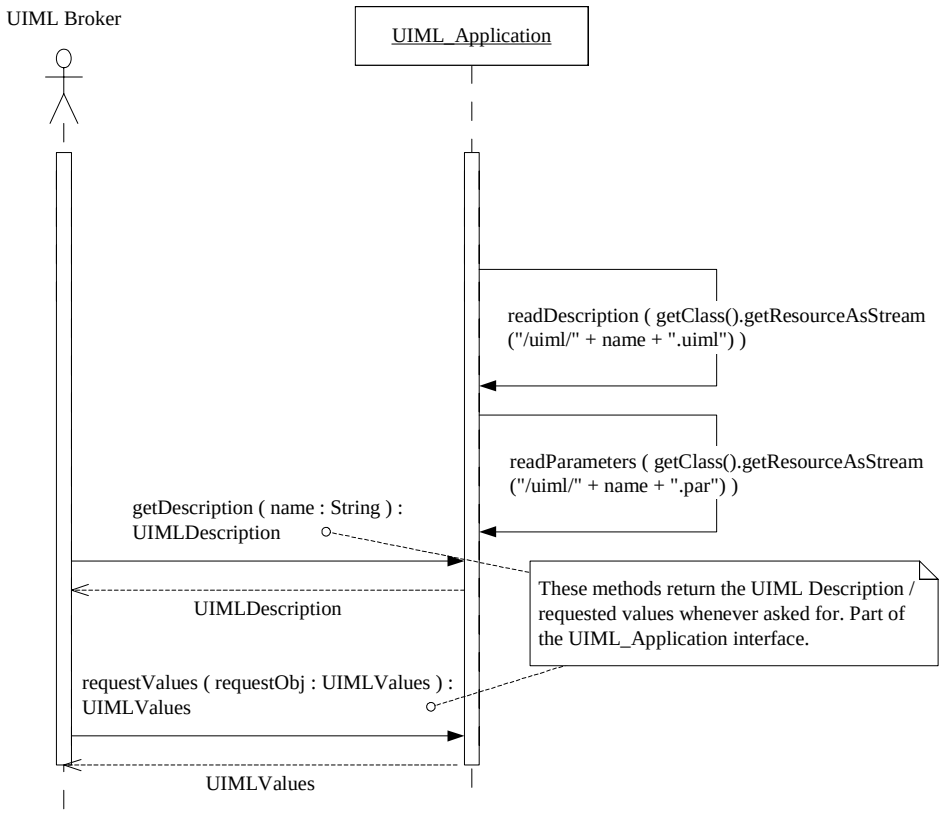

Figure 23. UIML Broker - Maintains UI Descriptions.

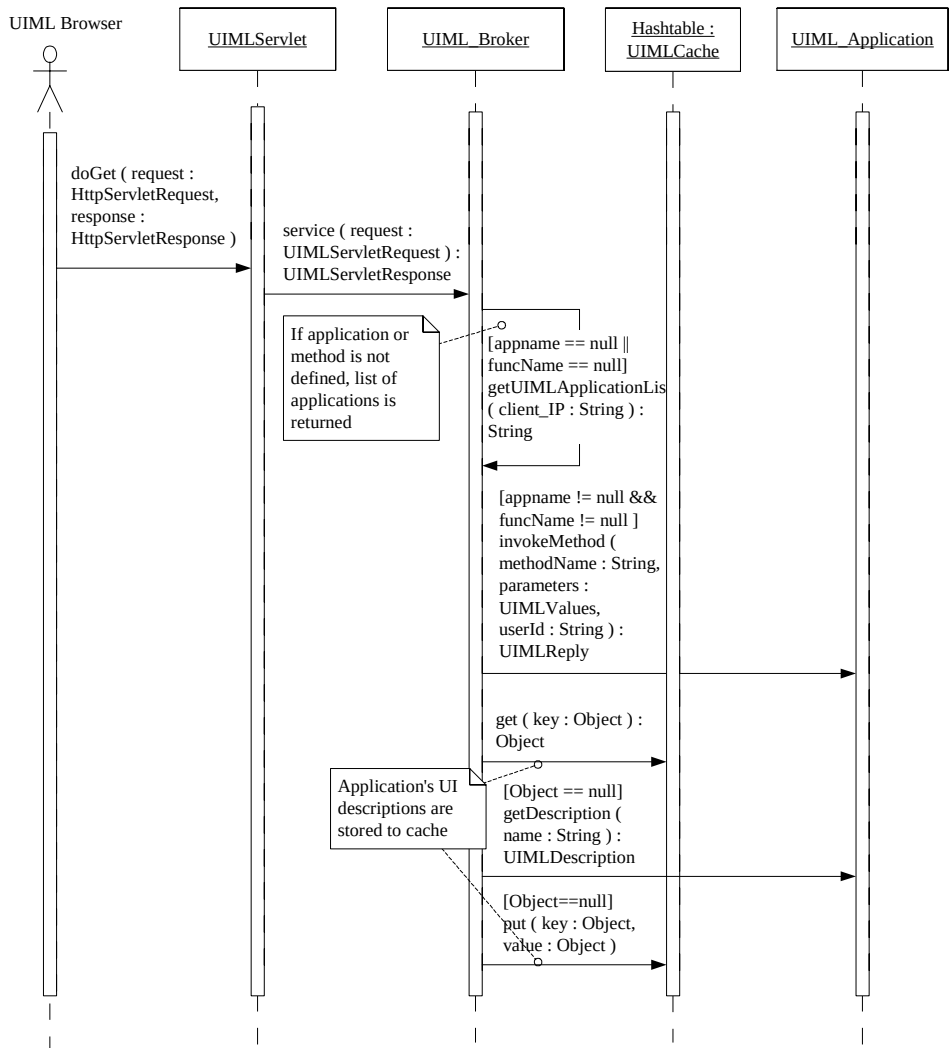

Figure 24. UIML Browser - Fetches and Updates UI.

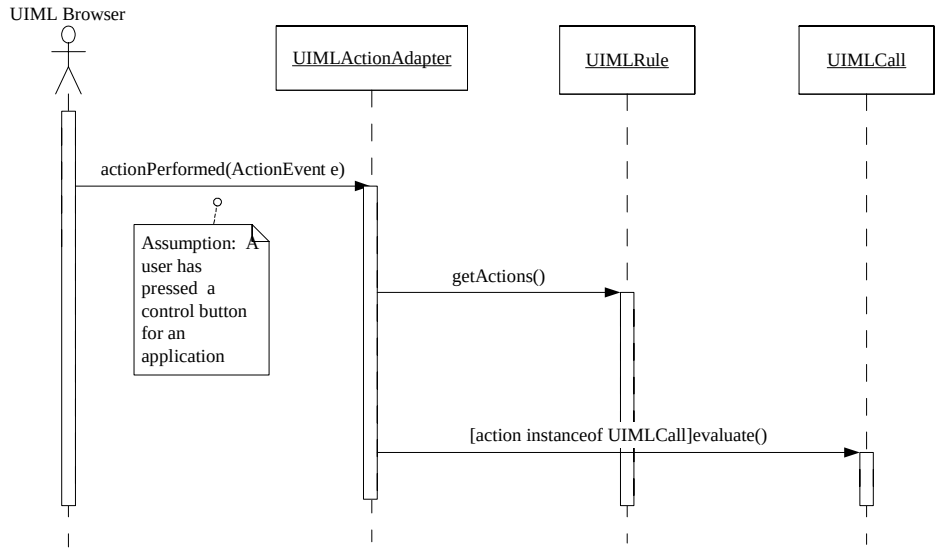

Figure 25. UIML Browser - Generates Control Events.

5.3 Chapter Summary

In this chapter, as entering **Step 3** introduced in Chapter 1.3, a case was analysed and a study was carried out to test the proposed method. The result, in the sense of system development, was a set of Java-specific interface definitions with a set of VHE-system objects needed to provide the functions that realisations of the UIML service will take.

This approach is prone to providing improvements for the architecture, API models and their implementations. Also, the approach exposes the inconsistency between the requirements and the implementation, and thus will show the parts of the system requiring further development.

Further development should comprise building of the implementation of the suggested APIs, bearing in mind the compatibility of the heterogeneous systems - the APIs should be built on the VHE Middleware APIs presented earlier in this text. This will verify the usefulness of the results, perhaps leading to further improvements of the UIML service APIs and VHE Middleware, and will eventually give a taste of real VHE.

6. Conclusions and Further Research

As the purpose of the document was said to be twofold, the conclusions will be twofold as well: 1) conclusions in the sense of scientific research and 2) conclusions in the sense of technical development of the VHE Middleware. Where we have reached and where we are going to will be covered. The question of the applicability of the proposed method will be evaluated and the qualities of the technical solution will be discussed. Furthermore, the issue of further R&D of VHE Middleware will be looked into. With the twofold starting point in mind, the results of the completed R&D will be analysed in this chapter.

6.1 Applicability of Proposed Method

An advanced method of identifying a conceptual architecture and APIs as a software system from its functional requirements was proposed. The proposed method is based on the common knowledge of OOA methodology, which claims that every software system contains hierarchical structures that reflect its functional requirements. In the OOA methodology, these structures and their hierarchy can be found by analysing and structuring the problem descriptions - the Use Case analysis is discussed. The proposed method here is to seamlessly move from a use case model to a conceptual architecture model of the software system. In the analysis, the UML's concept of extension point (expressing how a system might be extended) was deployed. Consequently, in the model, the layers and APIs of a software system were exposed. To evaluate the applicability of the proposed method, the three steps were completed. The steps and their results are as follows:

Step 1. **Method introduction** - This step is covered in Chapter 2. The proposed method was introduced in developing the Producer/Consumer paradigm. The hierarchy of evolving information systems, where a manually-driven information system was seen to be extended by automatic systems, was developed. The result was a functional decomposition model of the semi-automatic information system. In the semi-automatic information system the entities called the "Daemons" were helping the system to provide and consume services automatically, thus extending the manually-driven ones. In the analysis, the UML's concept of

extension point (expressing how a system might be extended) was deployed. Consequently, in the model, the layers and APIs of a software system were exposed.

Step 2. **Creation of a model of the problem** - This step is covered in Chapter 3. Using the proposed method in structuring the problem, an OO analysis was applied to a set of VHE Use Cases expressing the user's scenarios of the VHE services considered to be attractive to end users. These scenarios describe the functioning that is expected for fulfilling the idea of VHE presented by the VHE Middleware project. The analysis only dealt with three of the twelve VHE Use Cases authored by the VHE Middleware project. However, it can be assumed that a great deal of common functionality was found, and the findings will probably cover most of the functionality of the other nine VHE Use Cases. The result was expressed as an architecture concept of the VHE Middleware. The concept was modelled as an UML subsystem diagram and is illustrated in Figure 11. The results included the definitions of the VHE Middleware conceptual subsystems, their hierarchy, and the appropriate API definitions to deploy the concept.

Step 3. **Testing the applicability of the proposed method with a single case** - This step is covered in Chapters 4 and 5. To test the applicability of the proposed method, in Chapter 4 the defined APIs were modelled by applying pure OOA methodology to the problem descriptions presented by the VHE Use Cases. In Chapter 5 the methodology of Reverse Engineering was deployed to analyse the prototype software introduced in Appendix B. The functional decomposition model and API definitions developed in Step 2 were applied in the analysis of the prototype. Therefore, it was possible to compare the structures and APIs in the conceptual model with the structures and APIs found in the prototype software.

Conclusion 1: We have seen that the software functions and function groups revealed in the prototype software matched well with the theoretical model developed with the proposed method. For this reason, we can conclude that the proposed method is evaluated to be applicable.

6.2 Applicability of the Architecture

We have presented the design of the conceptual architecture with subsystems, layers and API definitions, and their further development, finally presenting the concrete interfaces of the VHE system components. By analysing the prototype software, we have proved that the architecture and its APIs will be useful and applicable when realisations of the VHE are to be implemented. Furthermore, in Section 1.1.4 it was found that the next three main attributes qualify the Middleware architecture, namely the *flexibility*, *extendibility* and *scalability*. How these attributes are realised in the developed architecture will be shown now.

The applicability of the developed VHE Middleware architecture can be evaluated by comparing it to the existing ones. In Chapter 1 the reference architecture models were introduced, namely the OMA, OSA, NMI, and OSGi. OSGi, however, will be discussed later on this chapter. Let us compare our architecture with these other three: The «core middleware» in Figure 26 corresponds to the "ORB" and "CORBA Services" presented with OMA in Figure 1, and to the "framework interface classes" presented with OSA in Figure 2, and, of course, to the "Core Middleware" presented with NMI. Respectively, the «upper middleware» in Figure 26 corresponds to the "Domain CORBA Facilities" and the "Horizontal CORBA Facilities" in Figure 1, and to the "network interface classes" of OSA in Figure 2, and to the "Upper Middleware" presented with NMI. The meaning of "Application" is the same in all architectures. In general, the purpose of software architecture is to illuminate the way in which the software system should be built. So far, we have presented a conceptual architecture; the conceptual architecture should be concretised to present a real architecture of some specific software system. In this case, the real one was the UIML Service, and we now have to illustrate it. This is done in Figure 26. The figure shows how the VHE Middleware architecture concept can be adapted to present the architecture of the UIML Service prototype system presented in Appendix B. By this, we will prove that our architecture is *flexible, extensible and scalable*.

To concretise the different possibilities for realising the architecture, we should refer to the several design models illustrated with the UIML Service case in Appendix B. The UIML Service is programmed on the OSGi framework. The

models can be easily mapped to the subsystem's realisation elements modelled as SM, DM, and AM of the architecture concept in Figure 11 and Figure 26. This mapping will be expressed when proceeding with this chapter. As another example of mapping the OSGi-based design model on the architecture developed by Eikerling et al. [28] should be noted. These design models can also be used to realise the architecture concept.

- **Realisation of the "VHE Middleware for Core Services"** - The OSGi System Bundle must be installed to enable the other Service Bundles to be installed. To be valid, a System Bundle, as every other service bundle in the OSGi environment, must implement the interface "Bundle". The interface Bundle here fulfils the purpose of the "Supports Automatic Service" API in Figure 26. In the OSGi runtime environment, the System Bundle is the element that provides the services called the "Core Middleware". To compare the System Bundle to the «core middleware» module of the architecture concept, we should look the six systems' state constants listed with the interface "Bundle" - namely "UNINSTALLED", "INSTALLED", "RESOLVED", "STARTING", "STARTED", "STOPPING", "STOPPED", "UPDATED", "MODIFIED", "REGISTERED", "UNREGISTERING" and "ACTIVE". These states fulfil the same purpose as that described in Figure 13. However, the collection of states in Figure 13 is much larger and slightly different to those in the Bundle. This is because the states in Figure 13 should serve all possible implementations of the «upper middleware» analysed in Chapter 2. Another difference is that the OSGi handles some of the states as events, causing state transitions in the system. This might be a better approach and should be considered in the future. However, the difference is slight when compared to the automatic state transitions, in which the system automatically enters from one state to the next without a specific event, except for leaving the previous state.

- **Realisation of the "VHE Middleware for GenericUI"** - In Figure 26, the UIML Service is drawn as a Bundle. It is application specific and is built to fulfil the purposes of the "Supports GenericUI Service" API and the "Supports Home Services" API. The OSGi offers the HTTP Service, which is here considered to fulfil the idea of a "Supports Service Usage" API - by using a standard HTTP browser a user can select a service on a list and start using the selected service. Examples of the realisation elements that point to

concretising the architecture are presented with the UIML Service case in Appendix B, in Figures 29 - 36.

- **Realisation of the "VHE Middleware for Remote Development" -** The OSGi's Device Access support is implemented at VTT Electronics so that the idea of a "Supports HA Devices" API is fulfilled - legacy devices can be deployed as home appliances. This is done by Aihkisalo et al. [29] in the project. Furthermore, the OSGi framework includes an HTTP Service to remotely administrate the OSGi environment installed on Home Server - the "Configuration Admin Service". This here fulfils the idea of a "Supports Remote Development" API.

6.3 Further Development

As with all software systems development, the development of architectures should be iterative, as should the VHE Middleware architecture development introduced in this publication. In addition to the OSGi-based UIML Service system, several prototype systems for different environments should be built, and, by this, the Middleware that is most flexible, extensible and scalable finally reached. In particular, the first set of interfaces to be standardised will have to be developed. These interfaces should be so general that they can be used far, far into the future, thus guaranteeing the compatibility of the different VHE systems to be realised in the future. Work will continue on this.

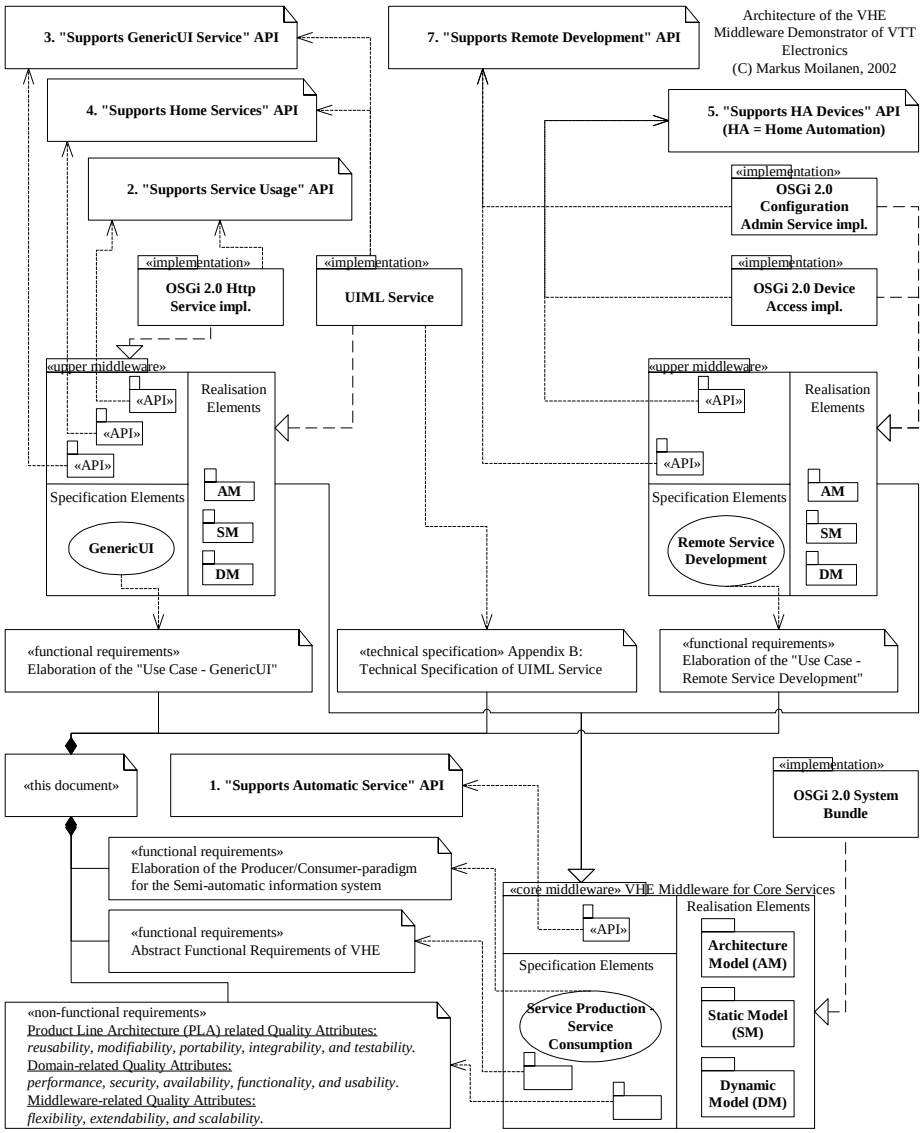

Figure 26. Architecture of the VHE Middleware Demonstrator of VTT Electronics.

Acknowledgements

The VHE Middleware project lasted over two years and the people working on the project at VTT Electronics have been involved as long. In addition to the author of the publication, and the other persons introduced in the Preface, we would like to acknowledge the following people for their contributions to the R&D of the VHE Middleware at VTT Electronics. They will be introduced in alphabetical order:

- **Tommi Aihkisalo**, Research Scientist at VTT Electronics. Participated in the development of the "Remote Service Development" user scenarios in Appendix A and the prototype development in Appendix B. E-mail: Tommi.Aihkisalo@vtt.fi.

- **Heikki Keränen**, Research Scientist at VTT Electronics. Participated in the R&D on the UIML Service in Appendix B. E-mail: Heikki.Keranen@vtt.fi.

- **Eila Niemelä**, Ph.D., Research Professor at VTT Electronics. Mrs. Niemelä is the author of the Abstract Requirements Specification document reproduced in Appendix C. E-mail: Eila.Niemela@vtt.fi.

- **Johan Plomp**, Licentiate of technology (electrical engineering) acts as senior researcher and a vice group manager at VTT Electronics. His role in the VHE Middleware project has been the co-ordination and initial realisation of the work on the GenericUI service concept. The research on adaptive and multi-modal interaction has benefited other simultaneous projects and will be continued in the Advanced Interactive Systems research field of VTT Electronics. His contribution to this publication has been mainly as originator and initial implementer of the GenericUI service concept, and the drafting of the scenarios in Appendix A. Contact via e-mail: Johan.Plomp@vtt.fi.

- **Tapani Rantakokko**, a student in Information Technology at University of Oulu and a trainee research scientist at VTT Electronics. Mr. Rantakokko has been working on the R&D of the VHE Middleware User Interface prototype system design and implementation. His contribution to this publication is in the chapters on "Case Study" and "Appendix B", both

handling the “UIML Service” prototype system implementation and analysis. Personal contact by e-mail to tapani.rantakokko@ee.oulu.fi.

- **Hannu Rytilä**, previously a Group Manager of the Software Design and Testing Research Group of Embedded Software Research Area at VTT Electronics. As Project Manager of the VHE Middleware project of VTT Electronics and a member of the VHE Middleware project Co-ordination Committee (PCC), Mr. Rytilä indirectly contributed to almost every issue of the VHE Middleware documentation. E-mail: hannu.rytila@nethawk.fi.

- A student group from the University of Oulu, Department of Information Processing Science, consisting of **Mika Hietala, Heidi Kallio, and Janne Suni**. These students carried out their Programming Project exercise by contributing to the API modelling in Chapter 4 and to the prototype analysis and modelling in Chapter 5.

References

1. VHE Middleware Consortium. *VHE Middleware Project WWW*. August 2002. URL: http:// www.vhe-middleware.org

2. Object Management Group. *Object Management Architecture*. August 2002. URL: http:// www.omg.org

3. International Standardisation Organisation. *Open Distributed Processing - Reference Model, X.901, ISO/IEC 10746*. 1995.

4. The 3rd Generation Partnership Project. *UMTS Technical Specification Group Services and System Aspects: Virtual Home Environment - Open Service Architecture 3G TS 123 127 V3.0.0*. August 2002. URL: http:// www.etsi.org/

5. National Science Foundation. *NSF Middleware Initiative*. August 2002. URL: http:// www.nsf.gov/pubs/2002/nsf02028/nsf02028.htm

6. OSGi Consortium. *Open Service Gateway initiative*. August 2002. URL: http:// www.osgi.org

7. Cockburn, A. *Writing Effective Use Cases. Addison-Wesley*. 2000. 270 p. ISBN 0201702258.

8. Büker, U., Eikerling, H-J. & Müller, W. *Proceedings International ITEA Workshop on Virtual Home Environments*. Organised by the VHE Middleware consortium, February 20–21, 2002, Paderborn, Germany. ISBN 3826598849.

9. Benbasat, I. & Goldstein, D.K. *The Case Research Strategy in Studies of Information Systems*. In: MIS Quarterly 11. September 1987. Pp. 369–388.

10. Järvinen, P. & Järvinen, A. *On Research Methods*. Tampere, Finland: Opinpaja Oy. 1999. 129 p. ISBN 951-97113-1-7.

11. Brown, W., Malveau, R., McCormick, H. & Mowbray, T. *Anti Patterns - Refactoring Software, Architectures, and Projects in Crisis*. Wiley Computer Publishing. 2002. 309 p. ISBN 0-471-19713-0.

12. Object Management Group. *UML V.1.4 Specifications*. August 2002. URL: http:// www.omg.org/ technology/documents/formal/uml.htm

13. UIML.org. *UIML Specification*. August 2002. URL: http://www.uiml.org/ index.php

14. Moilanen, M. *Architecture of VHE Middleware - Elaboration of Functional Requirements*. In: Proceedings International ITEA Workshop on Virtual Home Environment. Aachen: Shaker-Verlag. February 2002. Pp. 31–39.

15. Bass, L., Glements, P. & Kazman, R. *Software Architecture in Practice*. Addison-Wesley Pub Company. 1998. 452 p. ISBN 0201199300.

16. Lundberg, L., Bosch, J., Häggander, D., & Bengtsson, P-O. *Quality Attributes in Software Architecture Design*, In: Proceedings of the IASTED 3rd International Conference on Software Engineering and Applications. October 1999. Pp. 353–362.

17. Dobrica, L. & Niemelä, E. *A Survey on Software Architecture Analysis Methods*. In: IEEE Transactions on Software Engineering. Vol. 28, No. 7. July 2002. Pp. 638–653.

18. Dobrica, L. & Niemelä, E. *A strategy for analysing product line software architectures.* Espoo: VTT Technical Research Centre of Finland. 2000. (VTT Publications 427.) 124 p. ISBN 951-38-5598-8. URL: http:// www.inf.vtt.fi/pdf/

19. Weiss D., Lai, R. & Parnas. D. *Software Product-Line Engineering: A family-Based Software Development Process*. Addison-Wesley. 1999. 426 p. ISBN 0201694387.

20. Plomp, J., Mayora-Ibarra, O. & Yli-Nikkola, H. *Graphical and Speech-driven User Interface Generation from a single Source Format*. In: Proceedings of AVIOS 2001, San Jose, CA. 2–4 April 2000.

21. Plomp, J. *UIML in Future Home Environments*. In: Proceedings of the Aristote European Conference: UIML, a User Interface Mark-up Language. March 2001.

22. Burbeck, S. *Applications Programming in Smalltalk-80: How to use Model-View-Controller*. August 2002. URL: http:// st-www.cs.uiuc.edu/users/smarch/ st-docs/mvc.html

23. Sundsted, T. *MVC meets Swing. Explore the underpinnings of the JFC's Swing components*. August 2002. URL: http:// www.javaworld.com/ javaworld/ jw-04-1998/ jw-04-howto.html

24. Vaskivuo, T. *The Infrastructure of Interactive Devices in a Future Home*. In: Proceedings of the Dreaming for The Future - Future Home Conference, University of Art and Design. May 2001.

25. Büker, U. *Using Java Cards for Virtual Home Environments*. In: Proceedings International ITEA Workshop on Virtual Home Environments. Aachen: Shaker-Verlag. February 2002. Pp. 151–159.

26. Holappa, J. *Security threats and requirements for Java-based applications in the networked home environment*. Espoo: VTT Technical Research Centre of Finland. 2001. (VTT Publications 444.) 116 p. ISBN 951-38-5865-0. URL: http:// www.inf.vtt.fi/pdf/

27. Moilanen, M. *Management framework of distributed software objects and components*. Espoo: VTT Technical Research Centre of Finland. 2001. (VTT Publications 442.) 153 p + app. 45 p. ISBN 951-38-5863-4. URL: http:// www.inf.vtt.fi/pdf/

28. Eikerling, H-J., Buhe, G., Berger, F. & Görlich, J. *Architecting Middleware Components for the Control of Networked In-Home Multimedia Applications*. In: Proceedings International ITEA Workshop on Virtual Home Environments. Aachen: Shaker-Verlag, February 2002. Pp. 21–29.

29. Aihkisalo, T. Riihimäki, R. & Ranta, T. *Integration of Heterogeneous Home Automation Appliances: Generic Device Description Format*. In: Proceedings International ITEA Workshop on Virtual Home Environments. Aachen: Shaker-Verlag. February 2002. Pp. 13–20.

Appendix A: VHE Use Cases - Contribution of VTT Electronics

1. VHE Use Case - GenericUI

Owner: VTT Electronics

Scope: VHE

Originator: Johan Plomp

Status: Accepted by the PCC (Project Co-ordination Committee)

Change History

Issue	*Date*	*Handled by*	*Comments*
0.1 Draft	30 June 2000	Johan Plomp	First version circulated between partners
0.2 Draft	3 May 2001	Johan Plomp	Extended and changed format to Nokia template
1.0 Final	4 Sept 2002	PCC	The project is closed

Use case name (goal)

Generic User Interface Scenario - Register an appliance and convey the appliance UI description in a suitable format to a client used for controlling the appliance. See Figure 1.

System under discussion

A home environment, including a home server, some controllable devices (e.g. TV and VCR), a handheld control device and an internet connection.

Primary actor

The user of the system.

Level

System-level use case.

Supporting actors

- Handheld device (client)
- Appliance (may be more than one)
- Home server (also serving as a gateway)

Stakeholders

- Appliance manufacturers
- Home Server manufacturers
- Client device manufacturers

Pre-conditions

Home server up and running, with a possible connection to the internet

Availability of connections between home server and appliances, as well as client devices

Minimal guarantee

If the client can connect to server, list of controllable appliances (if any) will be passed to the client when the client UI format is supported. The appliance UI description will be rendered to a minimal set of client formats.

Success guarantee

Appliance-specific UI is rendered on the client device and can be used to control the appliance.

Main success scenario

1. Adding a new controllable device

When a device is added to the control area of a specific home server, the home server must be made aware of its existence and properties. This may be done automatically if the device is "VHE-enabled" and equipped with communication properties. Otherwise, the new device (e.g. a legacy device) must be added to the home server by manually adding its ID to the home server. This may be done via a control device or via the home server's user interface (if there is any).

2. Acquisition of the UI description

Upon addition of home equipment to the area controlled by the home server, the home server will attempt to retrieve a UI description for the control of the new device. If the device is a "VHE-enabled" device, it will be able to retrieve the UI description from the device itself. However, if the device is a legacy device or otherwise not able to provide its own description via a dedicated connection, the UI description must be retrieved from an external source. This external source may be located on the internet - e.g. a WWW or FTP service maintained by the manufacturer - or it may be provided with the device on a floppy, CD, or similar media.

3. Storage of the UI description

Maintaining the UI descriptions of the controllable devices is the task of the home server. It should have a description available for all devices that it can manage. If the description fails, there is no sense in displaying the device in the control list as a controllable device. Some of the control devices will be able to process the UI description as is, and may therefore want to maintain a copy of the UI description in their memory. This memory should work like a cache and

verify the version of the UI description regularly with the home server. The list of available devices should be handled likewise, and deleted devices must be removed from the local list as well. The home server may also take responsibility for checking for newer versions of the UI description by a regular check on the manufacturer's pages or by user-command.

4. Adding a control device

When a control device enters the control area of a home server, its list of controllable devices should be adjusted. For that purpose, the home server will send the list of controllable devices in a UI description-like form. The control device will merge that list into its own list (which may contain, e.g., remotely controllable devices at his home) and present the updated list to the user. The list will contain references to the UI descriptions available for the controllable devices.

5. Fetching a UI description

When a controllable device is selected from the list, the UI description for the device must be fetched. In case the control device maintains a cache of UI descriptions, this cache may first be checked for the availability of the controllable device's UI description. If it cannot be found from there, it must be retrieved from the home server. If it cannot be found there either, an error must be issued and the device removed from the list.

6. Transcoding the UI description

If the control device is not able to present the UI description as such by means of a dedicated browser, the UI description must be transcoded to a suitable format. This transcoding task is performed by the home server. The home server is aware of the client device's properties because of the handshake procedures it went through when the control device entered its control area. The home server will maintain a set of transcoding modules able to transcode the UI description format to other standardised formats. If a new format is requested, the server may download a transcoder from a given website. If necessary, the home server

may also implement a server for that specific format (e.g. HTTP for HTML or WAP for WML).

Note that if the transcoding is done in the device in order to show the UI by means of another application, from the system's point of view that device has a dedicated browser for the UI description.

7. Presenting the UI

The UI description in its original format will need a dedicated browser. It is very likely that such a browser can be built to run on a great variety of platforms (e.g. by implementing it in Java). However, some platforms will dictate other formats to be used, like HTML, WML and voice-based control (e.g. VoiceXML). Transcoding the UI description as described in the previous paragraph will resolve this problem, since the device's preferred format will be delivered to the device. Note that these formats may partly require that the home server implements or interacts with a server dedicated to these formats. An issue to address while choosing and utilising the UI description is the fact that the UI will be displayed on a variety of control devices with different properties. Most obvious are the difference in screen size (e.g. TV, Computer or PDA) and input devices (full keyboard vs. touch screen, pointing device or speech).

8. Passing control events

Operating the user interface by means of the control device will generate control events. These events must be passed to the device to be controlled. This can be done in two ways: either the events are channelled through the home server to the controlled device, or the device receives the control events directly from the control device. The chosen solution depends on the communication channels available and the capabilities of the devices. Channelling the events via the home server requires a connection from the control device to the home server, and from the home server to the controllable device. The control device and the home server will usually have a connection of some kind, but the second is not necessarily the case. For example, in the case of legacy devices like current TV

sets and VCRs, there is no link available between the home server and the device. However, both devices can properly be controlled via IR. A solution in this case could be to use the IR port of the control device (if available) to control these devices. This requires a conversion of the events to IR port signals, and possibly a special driver for the device.

Unresolved issues

Still unresolved issues include security matters and authentication. It is expected that solutions to these issues will partly come from the Middleware side.

Related to the issue of security is the conflict that may result from two users trying the control the same device. Mechanisms to resolve this must be found.

A third problem related to this is the difference between local and remote interfaces. It is not desirable or advisable to provide all the controls of a local control - i.e. used in the same room - on a remote control - i.e. used while being out of the house. One will only want to change the channels on a TV set when at home and watching the television, not when not even being able to see the TV set. On the other hand, programming a VCR to record a program should also be possible remotely.

An important feature of the whole VHE concept is its ability to adapt to the user. The user's preferences must be taken into account in, e.g., the look and feel of the UI, the services provided or ranked highly, and the functionality of the services.

Exceptions

List here the main exceptions to all steps. Write the exceptions to step 3 as 3a, 3b, 3c, etc. Write the repair steps to exception 3a as 3a1, 3a2, 3a3, etc. Indent as explained below.

Exceptions of step 1 1a Connection with new appliance cannot be established 1a1 Add appliance manually
Exceptions of step 2 2a UI Description cannot be fetched from device (not VHE enabled, or connection down) 2a1 Ask for description location from user (first use) 2a2 Download from the internet or local media 2a3 Use description from cache (later uses)
Exceptions of step 3 3a No space for new description 3a1 Use description and throw away (needs to be fetched for each use) 3b Description in storage (cache) out of date 3b1 Fetch new description 3c Appliance removed from environment / connection cannot be established 3c1 Remove appliance from list
Exceptions of step 4 4a Client (control device) cannot connect to server 4a1 Signal error message to user (client's responsibility) 4b Server does not support client's format 4b1 Signal error message to user (how?)
Exceptions of step 5 5a UI description cannot be found in cache of client or home server, neither can it be retrieved from the appliance 5a1 Issue error to user
Exceptions of step 6 6a Suitable converter for client's format is not available 6a1 Issue error message to user

6b UI Description does not conform to the standard or conversion generates an exception 6b1 Issue error message to user
Exceptions of step 7 7a Generated UI description cannot be rendered by the client 7a1 Client issues an error message to the user
Exceptions of step 1 8a Connection with appliance cannot be established (neither through home server) 8a1 Issue an error message to the user
Sub use cases
Release: v.1.0
Status: Accepted
Author: Johan Plomp

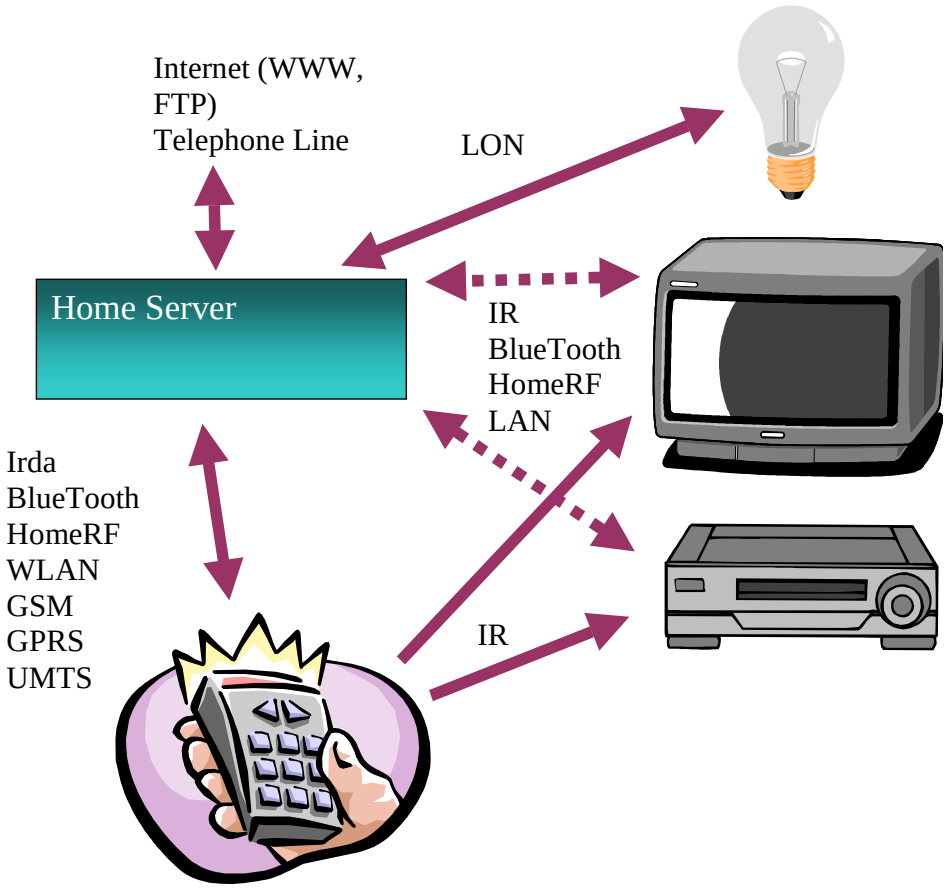

Figure 1. GeneriUI.

2. VHE Use Case - Remote Service Development

Owner:	VTT Electronics
Scope:	VHE
Originator:	Tommi Aihkisalo
Status:	Accepted by the PCC

Change History

Issue	*Date*	*Handled by*	*Comments*
0.1 Draft	18 May 2001	Tommi Aihkisalo	First version
1.0 Final	4 Sept 2002	PCC	The project is closed

Use case name (goal)

Remote Service Development – Remote development and maintenance of home automation services. See Figure 2.

System under discussion

A home environment, including a networked home server and some controllable home automation devices.

Primary actor

The service provider offering remote service development.

Level

System-level use case. Connections are assumed to exist, not relevant to the use case.

Supporting actors

- Automation platform with several automation devices.
- Home server (also serving as a gateway)

Stakeholders

- Appliance manufacturers
- Home Server manufacturers

Pre-conditions

Home server up and running, with a connection to the internet

- Availability of connections between home server and appliances. There is a sufficient amount of description information available about the automation platform in usable form

Minimal guarantee

The remote developer gets a connection to the home server and receives an information bundle concerning the devices in the automation platform.

Success guarantee

All the devices in the automation platform are described and it is delivered to the remote developer to develop a service application to that specific platform. Furthermore, the new application is delivered to the home server.

Main success scenario

1. Gathering information from the devices in automation platform

When a device is added to the control area of a specific home server, the home server must be made aware of its existence and properties. The home server collects all the information available from the new device. The information can be partly collected from the device itself and be completed with information from an external source, e.g. a suitable database or a resource collection. A suitable API for the device is needed.

2. Collecting information about current applications

The applications currently running are described in a suitable format. The information can be collected from independent devices and/or external sources in the home server.

3. Preparing for a contact from remote service developer

The home server completes both the automation platform and application description in a suitable format e.g. utilising XML. The availability of information is announced to a registry mechanism, which updates versioning information and the existence of such descriptions. The remote developer is also informed that changes have occurred in the automation hardware.

4. Fetching a hardware platform and application information

The remote service developer contacts the home server over the Internet and looks for the newest automation platform and application information from the registry mechanism. if necessary, the information is transmitted over the network to the developer.

5. Developing the application

The developer can develop a suitable service application in an off-line state in

his office, away from the user's home hardware in question. The development is easy because the developer retrieves all the properties of that hardware platform from the descriptions. At the same time, the UIs are updated or new ones created.

Unresolved issues

- Still unresolved issues include security matters and authentication. It is expected that solutions to these issues will partly come from the Middleware side
- Not all the devices embed self-descriptive information, even for identification
- The availability of suitable APIs for all devices
- The lack of a common interface for all home devices
- The lack of the service developer's tools

Exceptions

List here the main exceptions to all steps. Write the exceptions to step 3 as 3a, 3b, 3c, etc. Write the repair steps to exception 3a as 3a1, 3a2, 3a3, etc. Indent as explained below.

Exceptions of step 1

1a The device description cannot be fetched from the device

1a1 Ask for description location from user (first use)

1a2 Download from the internet or local media

1a3 Use description from cache (later uses)

Exceptions of step 4

4a The remote developer cannot connect to home server

4a1 Signal error message

Exceptions of step 5

5a In on-line development state, the connection to home server breaks

5a1 Issue error to user

5a2 Continue developing on the basis of local copy

Exceptions of step 6 6a Connection to home server cannot be established 6a1 Start a network agent that transmits the new application when possible.
Exceptions of step 7 7a The new application cannot be started due to an error in the application 7a1 Inform the remote developer and describe the problem
Sub use cases
Release: v.1.0
Status: Accepted
Author: Tommi Aihkisalo

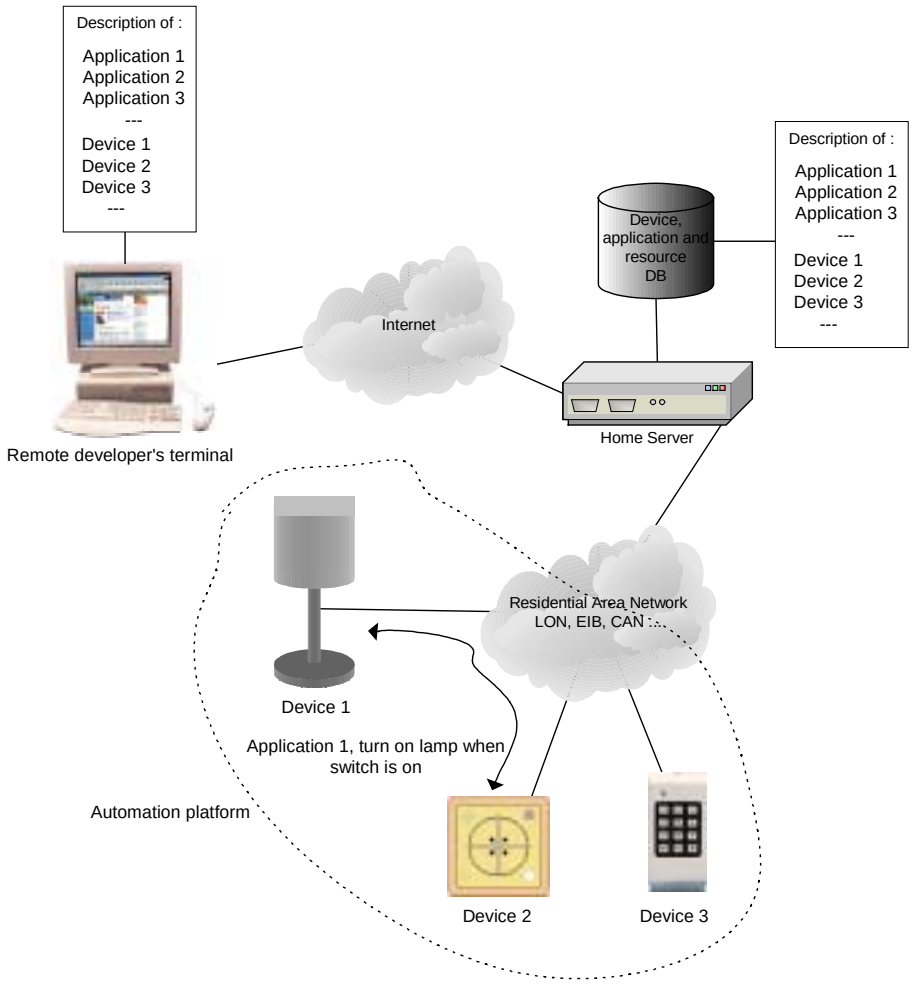

Figure 2. Remote Service Development.

3. VHE Use Case - Serverless Service Provisioning

Owner:	VTT Electronics
Scope:	VHE
Originator:	Hannu Rytilä
Status:	Accepted by the PCC

Change History

Issue	*Date*	*Handled by*	*Comments*
0.1 Draft	4 May 2001	Hannu Rytilä	First version circulated between partners
1.0 Final	4 Sept 2002	PCC	The project is closed

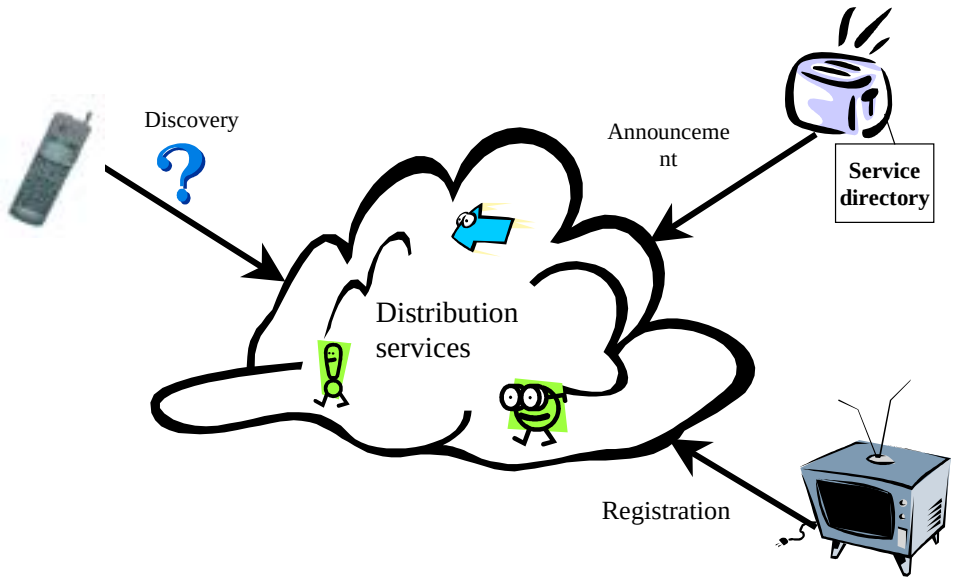

Figure 3. Serverless Service Provisioning.

Use case name (goal) Serverless Service Provisioning – Interoperability of two VHE aware devices when a special VHE server or service gateway is not reachable. See Figure 3.
System under discussion At least two VHE aware devices, which have been taken out from the primary home environment.
Primary actor The user of the system
Level System-level use case. Connections are assumed to exist, not relevant to the use case.
Supporting actors • Client (handheld terminal, which does not provide services) • Appliance (may be more than one)
Stakeholders • Appliance manufacturers • Terminal manufacturers
Pre-conditions All devices are switched off and there is no home server reachable by any supported communication channel.
Minimal guarantee The appliance is able to make its services available and the client (a terminal or another appliance) is able to find the list and identify the services.
Success guarantee The client is able to use services of the appliance.
Main success scenario

1. The appliance is turned on

The appliance initiates all communication channels that are allowed and supported in the current context.

2. Service discovery

The appliance goes through a hierarchical service discovery through all active communication channels in order to find the VHE-compliant directory service. The discovery fails because no directory services are available.

3. Initiation of service directory

The appliance initiates directory service, which at least has enough space for its own services. The appliance starts announcing the existence of directory service through all active communication channels.

4. Service registration

The appliance initiates its services one by one and registers them into its own service directory.

5. Another appliance is turned on

The other appliance goes though steps 1 and 2, but finds the service directory of appliance 1 and registers its services into it.

6.The client is tuned on

The client initiates all communication channels that are allowed and supported in the current context.

7. Service discovery of the client

The client goes through a hierarchical service discovery through all active communication channels and identifies the directory service initiated by the appliance.

8. Retrieval of the service list

The client requests a service list from the directory service and retrieves the list in an appropriate format. The terminal informs the user that there are services

available.

9. Service usage

By request, the client displays the service list for the user, who activates one of the services provided by the appliance.

Unresolved issues

Still unresolved issues include security matters and authentication. It is expected that solutions to these issues will partly come from the Middleware side.

Exceptions

List here the main exceptions to all steps. Write the exceptions to step 3 as 3a, 3b, 3c, etc. Write the repair steps to exception 3a as 3a1, 3a2, 3a3, etc. Indent as explained below.

Exceptions of step 3

3a the appliance is not capable of running directory service

3a1 The appliance remains listening to see if a service directory comes available.

3a2 The terminal has to initiate a service directory where appliances can register their services

Exceptions of step 5

5a the service directory of appliance 1 does not have space for the services of appliance 2

5a1 Allow multiple service directories and supply mechanisms to join them in client side.

5a2 Appliance 2 initiates its own service directory and registers it into the service directory of appliance 1

Sub use cases

Release: v.1.0

Status: Accepted

Author: Hannu Rytilä

Appendix B: Technical Specification of UIML Service

Owner: VTT Electronics

Scope: VHE

Originator: Tapani Rantakokko

Status: Accepted by PCC

Change History

Issue	*Date*	*Handled by*	*Comments*
0.1 Draft	30 June 2000	Tapani Rantakokko	Document is created
1.0 Final	28 June 2002	Tapani Rantakokko	First version circulated between the VHE Middleware project's partners

1. Purpose

The purpose of this document is to briefly describe the general architecture of the Distributed Generic User Interface Platform implementation, which we will call the "UIML Service", and to present its internal and application side interfaces in detail for the service and application developers.

The system implementation uses XML-based UIML as the language for describing user interfaces [1], and, therefore, forms the basis of the UI technology presented in this text. The current implementation of the UIML Service is built according to the OSGi 2.0 service platform specification (Open Services Gateway initiative) [2], which is a third-party implementation of a system framework. Both are out of the scope of this text and will only be referred to. No knowledge of these is required in order to understand this text, but it is clearly helpful.

2. System Specification

The number of devices that can be used to control home equipment is rapidly increasing. Mobile phones, portable Internet terminals, handheld computers and communicators are examples of devices that already hold the required capabilities. A set of these control devices is presented in Figure 1. As these control devices typically use different languages to describe user interfaces, traditional programming forces us to completely re-code each application's UI whenever a new type of controlling device is added to the system. Voice-based interaction is another, natural way of controlling devices, but it also sets its own requirements on UIs.

Figure 1. Different types of controlling devices and user interfaces.

UIML is currently under development and strives to solve this significant problem of re-coding UIs by providing XML-based descriptive language which can be automatically transcoded to other formats, like HTML, JAVA or even languages not yet invented. VTT Electronics is participating in the development of UIML, keeping one-file one-time generic user interface description technology as its aim. To achieve this goal, we have developed extensions to the UIML language, a generic vocabulary and a transcoding tool.

In home environments, services are typically built on top of a specific home server by way of networking. This server is also connected to a wide-area network in order to provide remote control. OSGi 2.0 describes the delivery of multiple services over wide-area networks to local networks and devices. To provide a flexible control system for home environments, we have now combined our user interface technology with the OSGi approach and, therefore, gained a distributed generic UI platform.

2.1 Partitioning

The distributed generic UI platform includes three parties: the *controllable devices*, *control devices*, and the *enabling service* between them, which is here called the “UI Broker” (see Figure 2) The UI Broker concept is a generalisation of the UIML Service system, including all the components needed to provide communication between controllable and control devices.

In this UI platform, controllable devices do not have to support different types of UI description languages but only one, the UIML language, which is based on XML and is formally easy to learn. However, it should be noted that creating generic user interfaces requires careful designing of the UI components and the logic behind them.

When the developers build software supplied with UIML descriptions of the user interfaces, these applications can be made available to a UI Broker, so they can be later accessed through references it handles. In addition, an application list can be shown. When the user chooses an application from the list, the UI Broker then contacts it and transcodes its UIML-formatted UI description to a format that a current control device can accept. User profiles can also be used, for example to give applications user-specific outlook and/or behaviour.

After retrieving the requested application's UI, the user can interact with the application through the UI in a traditional fashion. Behind the curtains, the user's actions on the UI will be converted to the UIML format and passed to the application backend through the UI Broker. After processing the action, a new, possibly updated, UI description will be returned.

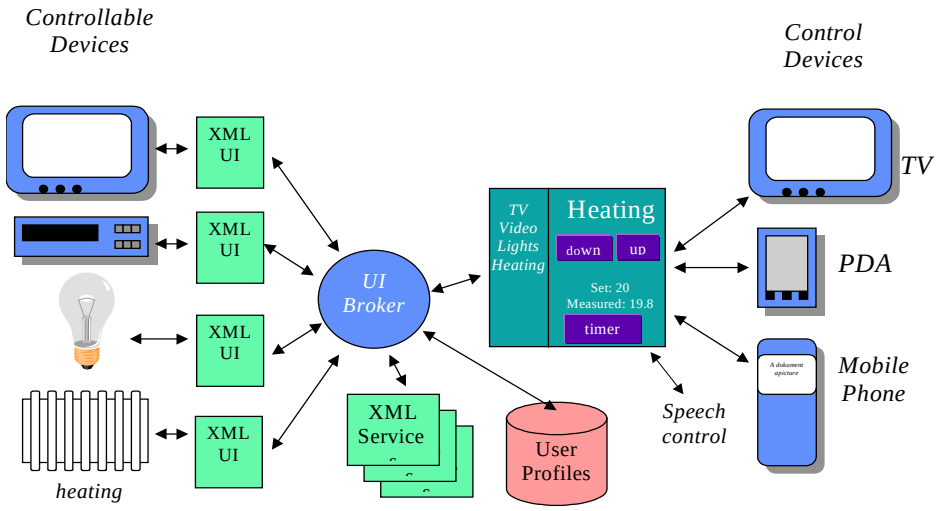

Figure 2. Using a generic user interface description for controlling devices. Illustration: Johan Plomp.

In our approach, the distributed generic UI system contains three main interfaces and implementing classes: the *UIMLService*, *UIMLTranscoders*, and *UIMLApplications*, illustrated in Figure 3.

The heart of the UI Broker is the UIMLService component, which is implemented as a service that other programs running inside the system framework can use. It communicates with applications and transcoders internally through specified interfaces, and with client devices through an external HTTP server. UIMLService co-ordinates the necessary transcoding process between parties by exploiting the capabilities of UIMLTranscoders.

When running, the UIMLService component automatically searches UIML - capable applications from the system framework. Found matches are provided as a list of application names with short descriptions. When the user chooses an application, the UIMLService contacts it and then handles the communication between the UI running on the control device and the application backend, which is possibly controlling an external device.

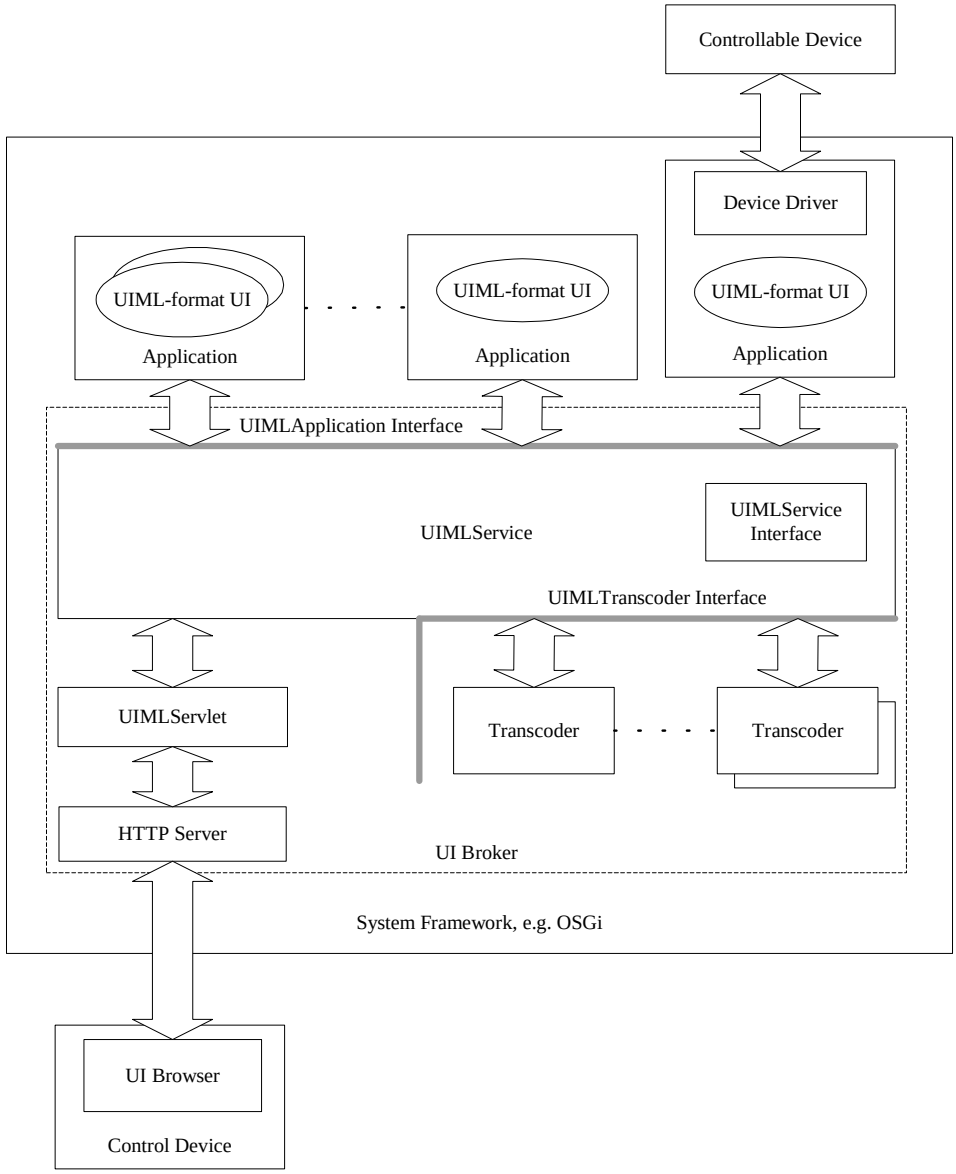

Figure 3. UI Broker implementation.

In the current implementation of UI Broker, control devices use an HTTP server in communication with UIMLService. A servlet, which is called the "UIMLServlet", is provided in order to make this kind of communication possible. The UIMLServlet's main purpose is to pass an HTTP server's requests to the UIMLService and receive responses back, but it also handles client format recognition by utilising the information included in the HTTP's requests'

headers. Currently, the most typical HTML and VXML browsers are recognised, and the output will be automatically transcoded to the right format. In other cases, the output format can be set externally by using a specific format parameter in the URI call (Universal Resource Identifier), e.g. format=HTML. For further details of the functionality provided by UIMLService, see the interface “UIMLService” in Section 2.4.

UIMLTranscoders are packages providing resources for UIMLService, each including one or more transcoders from the UIML to the destination languages. When the UIML language evolves, or new transcoders for new target formats are developed, transcoders can be updated or added to the system separately. A suitable transcoder will be automatically found and used by UIMLService whenever required, if such a transcoder is properly installed in the system. This means that a transcoder must register itself into the framework as a service that implements the interface “UIMLTranscoder” and provide the identification information (see Section 2.5). The required functionality of a transcoder for the UI Broker system is described in detail in the documentation of the interface “UIMLTranscoder” in Section 2.4.

The application side connections of UIMLService are created through the system framework, in this case through the OSGi-service technology. In order to communicate with the UI Broker, applications must register themselves as services that implement the interface “UIMLApplication”, and provide the identification information of each (see Section 2.5). The required functionality of an application for a UI Broker system is described in detail in the UIMLApplication interface (see Section 2.4).

2.2 Message Sequences

The UI Broker component’s interaction against time is illustrated in the following figures using the UML notation.

When the system framework is first started (see Figure 4) the HTTPserver registers itself into the framework. It also initialises UIMLService, the transcoders and the applications. Next, the UIMLServlet and 1–2 of Push Servers threads will be started. The servlet and the Push Client's applet will be

registered into the HTTP server's namespace with other provided resources like the UI images of UIMLService. Finally, the UI Broker is up and running.

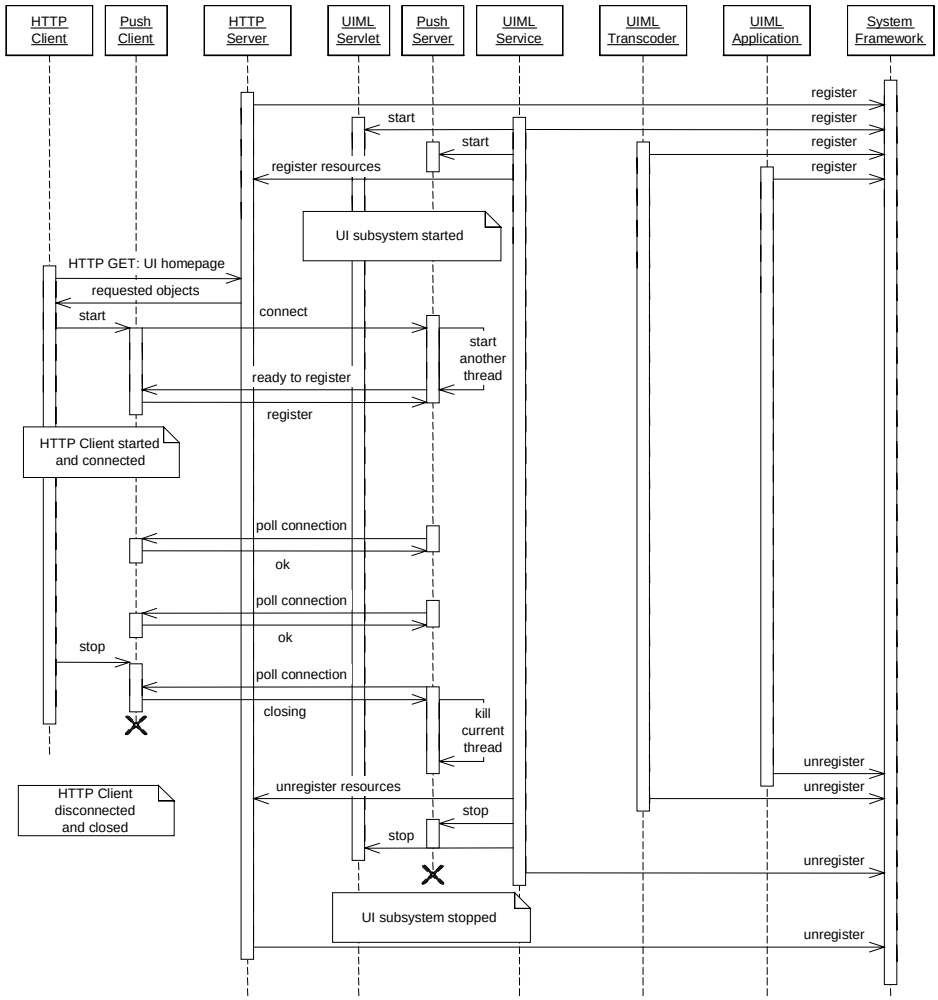

Figure 4. Message sequence of UI Broker start-up / shutdown.

After the UI Broker has started, HTTP clients may contact the URI the UIMLServlet has registered to, and may request a list of applications. If a push service is to be used with an HTML browser, a special URI specified in the UIMLService settings should be used instead, and a frameset including the push applet will be loaded to the browser.

If the push service is used, the Push Client will now contact through TCP to a Push Server thread running inside the UIMLService. A new server thread will be started in the Push Server to ensure that the next client can also be served. When the handshaking with the Push Client is finalised, the connection is ready for transmitting push commands. Every now and then, each Push Server thread will poll its connection to the client to see if it's still alive. If not, the connection will be closed and the server thread will be killed. If the Push Service is not used, none of this will happen. When the system is shutting down, all components will be unregistered and stopped.

If an HTTP client device contacts the UI Broker's HTTP server and the UIMLServlet attached to it, an application list should be created in the UIML format and returned to the client, if no other application is specified in the request (see Figure 5).

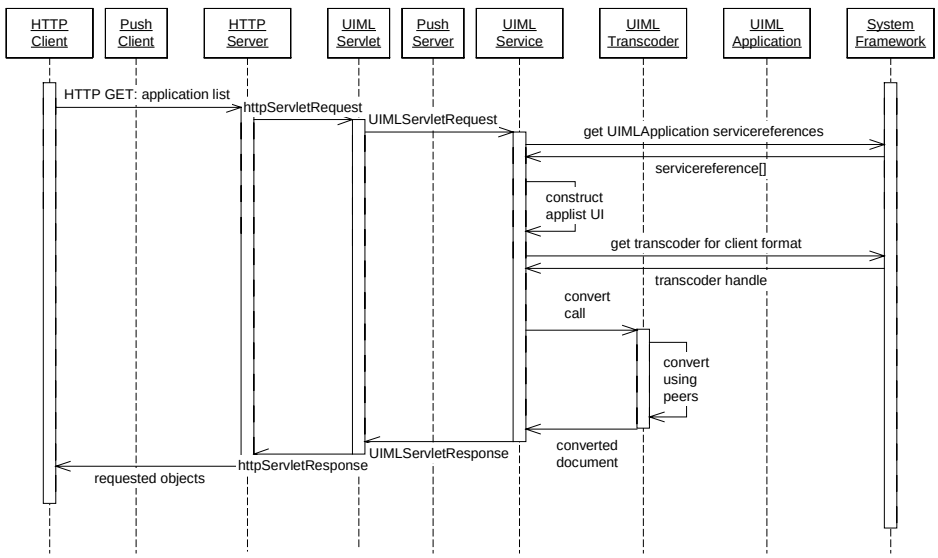

Figure 5. Message sequence of acquiring the application list.

When a request arrives at the UIMLService component, it will first check the parameters of the request and, if no valid application and method name is found, will acquire a list of running UIMLApplications from the framework they have been registered to and construct a list of applications in the UIML format. Based on the client format detection, a decision will be made as to whether it is necessary to transcode the application list to UI. If the answer is yes, the

UIMLService will begin searching for a suitable transcoder from the framework. If a transcoder that can handle the current client format is found, the UI will be put through it before returning to the client.

After acquiring a list of applications, the user is now able to choose a service to be used. When selected, a request should be sent to the UIMLService and a UI of the selected service should be returned (see Figure 6). The user may then start to interact with the application through the newly returned UI, and similar requests will be sent to the UIMLService again.

In this procedure, the UIMLService will always first try to find the requested application from the framework, and the search is based on the name of the application. When found, the name of the method in the request will be checked. If it is a direct request for a named UI, the UI description will be acquired immediately using the getDescription() method specified in the interface "UIMLApplication". This is the case when, for example, the call originates from the application list.

Otherwise, an invokeMethod() call is placed first and the reply from the application is interpreted. Based on this reply, a decision is made as to whether it is necessary to get a new UI description from the application or if the cache version can be used. In the latter case, the UIMLService will try to find a matching UIML-format description from the cache instead of requesting it from the application. If not found, the UI has to be acquired from the application.

The transcoding process is somewhat similar to the one described in Figure 5, but now caching of transcoded UIs is supported. In other words, the cache in UIMLService handles both pure UIML-format descriptions and already transcoded descriptions. In addition, acquiring values from the application during the transcoding process is possible through the interface "UIMLService". Finally, the descriptions (pure UIML and transcoded version) are stored in the cache and the client format description is returned to the client.

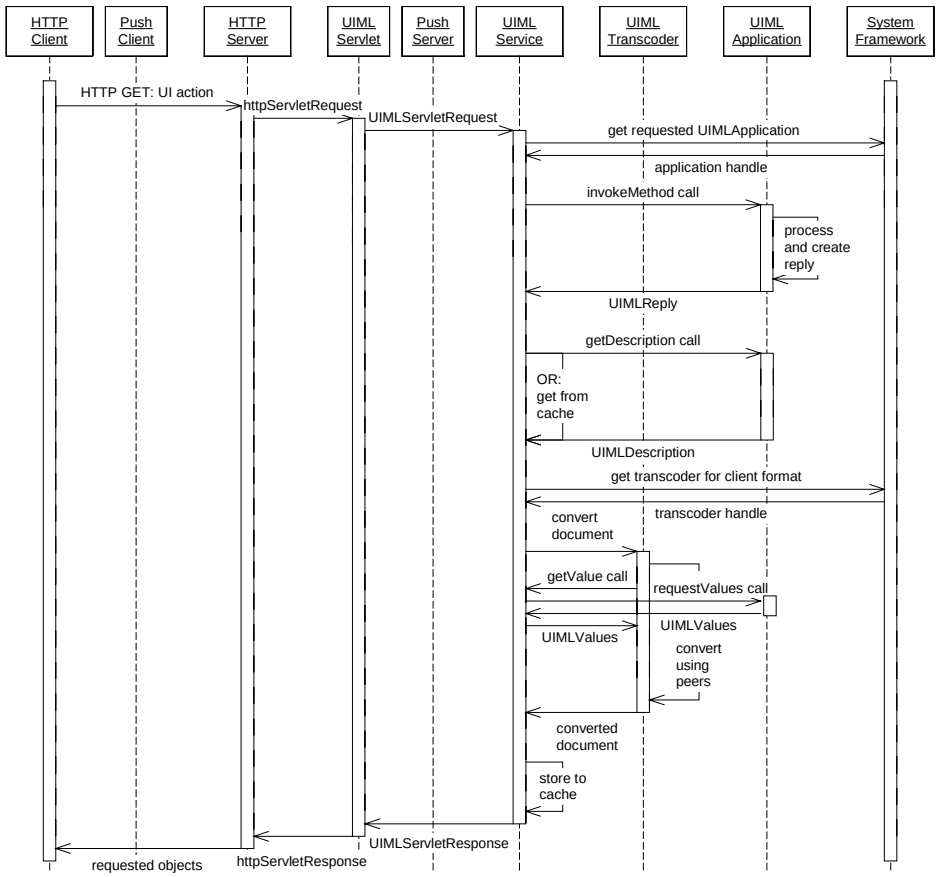

Figure 6. Message sequence of communication with applications.

A newly added feature in UIMLService is the support for application-initiated updates, called the “Push Service”. This means that applications that wish to update their UIs seen on the client devices can now initiate the update process whenever required, whereas earlier UIs were only updated when the user created some action first. However, due to the limitations of the HTTP protocol and target formats, this capability may not be available for all clients and UIs should be designed to work properly without the Push Service.

The push process is illustrated in Figure 7. The interface “UIMLApplication” includes methods for enabling and disabling the push service; these are to be used to inform applications whether it is reasonable to send push messages or not. When the push service is in use, the applications can push UI update

messages to the UIMLService through the interface “UIMLService”, and the UI Broker will handle the rest of the process.

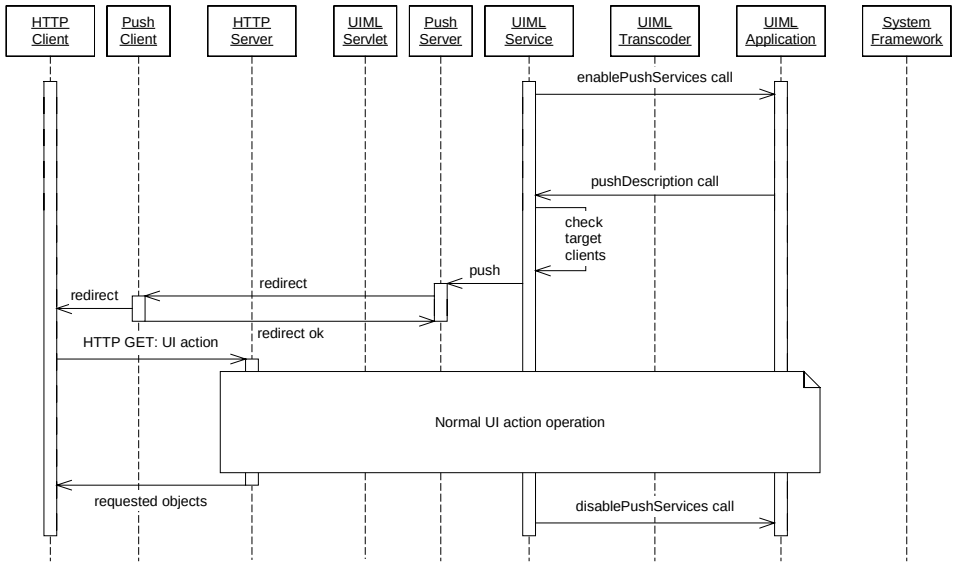

Figure 7. Message sequence of push service usage.

In practice, UIMLService has a live connection to all push-enabled clients and has a registry including information on which specific UI each client is currently viewing. When an UIMLService receives a push message from an application, it searches through this registry and sends an update message to all clients that are currently viewing the pushed description. The application does not send the description itself in the push method, but a command including the name of the application and the id of the UI description. The update process will be handled just like a normal UI action request.

2.3 Packaging

In Figure 8, the packaging of the UI-system is illustrated using the UML notation.

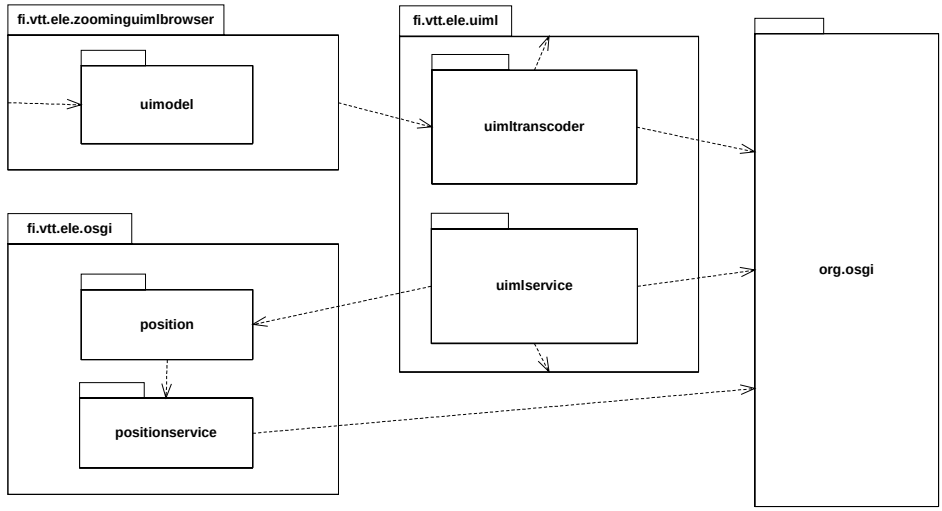

Figure 8. The UI system package diagram with dependencies.

Common classes and interfaces for the UI components can be found from the fi.vtt.ele.uiml package and the UIML service, with a reference implementation of the UIMLTranscoder also located inside this package. The system framework, in this case the OSGi 2.0 implementation, can be found in the org.osgi package. A browser capable of transcoding pure UIML-format UIs to Java UIs at runtime uses the UIMLTranscoder reference implementation for transcoding the UIML and thus has a dependency on the UIML transcoder package. This browser is an additional part of the UI Broker system implementing a multi-platform portable UIML browser, and can be found in the fi.vtt.ele.uiml.zoom in UIML browser package. Positioning packages found from the fi.vtt.ele.osgi are used by the UIMLService to modify its behaviour based on the positions of other devices, like the current control device. In the current implementation, the application list varies according to the client devices' locations.

As the UIMLService is a part of VTT Electronic's ITEA-VHE demonstrator, several components have been added to the whole demonstrator, including TV/VCR (TV-set/Video Cassette Recorder) controlling through infrared, WLAN positioning (Wireless Local Area Network), home automation through X10 and LON (Local Area Network), Bluetooth, and the IPv6-solutions, just to mention a few. The UIs for these components have been successfully implemented using the UIMLService. See the additional illustration of the UIML Service in Section 2.6:

- Figure 9. VHE-demonstrator’s package diagram.
- Figure 10. VHE-demonstrator’s deployment diagram.
- Figure 11. VHE-demonstrator’s OSGi-server component diagram.
- Figure 12. Screen captures of the UIML UIs.

2.4 Interface Specification

Three interfaces are provided: the “UIMLService”, “UIMLTranscoder” and “UIMLApplication”.

The services offered by the UIMLService component are described in the interface “UIMLService”. This interface includes two methods: one for pushing the UI descriptions (used by the UIMLApplications) and one for acquiring values from the UIMLApplications (used by the UIMLTranscoders during the transcoding phase).

The most important interface from an application developer's point of view is the interface “UIMLApplication”. This describes the methods that should be implemented by an UIML application; it contains methods for retrieving the UIML-formatted UI descriptions, for requesting values, for invoking methods, and for enabling or disabling a Push Service.

In addition, three helper classes are provided: the “UIMLDescription”, “UIMLReply” and “UIMLValues”.

The UIMLDescription class represents the UIML-formatted UI description with identification information. The UIMLReply class represents the UIMLApplication's reply to the invokeMethod() call specified in the interface “UIMLApplication”.

The UIMLValues class is used for passing values between the UIMLService and the UIMLApplications.

Compiled Java runtimes of these public interfaces and helper classes can be found in the fi.vtt.ele.uiml package.

2.5 Requirements for Services

Applications - In order to be found by UIMLService, applications must fulfil three requirements as follows:

1. The application must implement the interface “UIMLApplication” properly.

2. The application must be registered to the system framework as a service that implements the interface “UIMLApplication”.

3. When registering to the framework, the application must provide 'name' and 'desc' values. These will be used as identification keys for different applications, and in the process of creating the application list. The name of the application must be unique, just to ensure that applications can be distinguished. Because of this requirement, it is a good practice to add the manufacturer's id to the beginning of the name. It is also recommend that the description of the application is unique, though this is not required. The description may be shown to the users, the name will only be used internally.

Of course, the application must also be 100% UIML and OSGi 2.0 compliant. In addition, the UIML descriptions have to follow the requirements set by the extensions to the UIML language created by VTT Electronics.

In this text, neither the writing of UIML-format UIs nor building the OSGi bundles is discussed further. An example of a UIMLApplication is included in the UI Broker deployment package.

Transcoders - In order to be found by UIMLService, transcoders must fulfil three requirements as follows:

1. The transcoder must implement the interface “UIMLTranscoder” properly.

2. The transcoder must be registered to the system framework as a service that implements the interface “UIMLTranscoder”.

3. When registering to the framework, the transcoder must provide 'name' and 'desc' values. These will be used as identification keys for different transcoders, and in the process of acquiring a suitable transcoder. The name of the transcoder must be unique, just to ensure that the transcoders can be distinguished. Because of this requirement, it is a good practice to add the manufacturer's id to the beginning of the name. It is also recommend that the description of the transcoder is unique, though this is not required. The description may be shown to the users, the name will be used only internally. In addition, the transcoder must provide a list of target languages that it can handle in a value named the “codecs”, e.g. HTML or VXML. UIMLService will choose the transcoder to be used by comparing this value with the available UIMLTranscoders.

2.6 Additional Illustrations of UIML Service

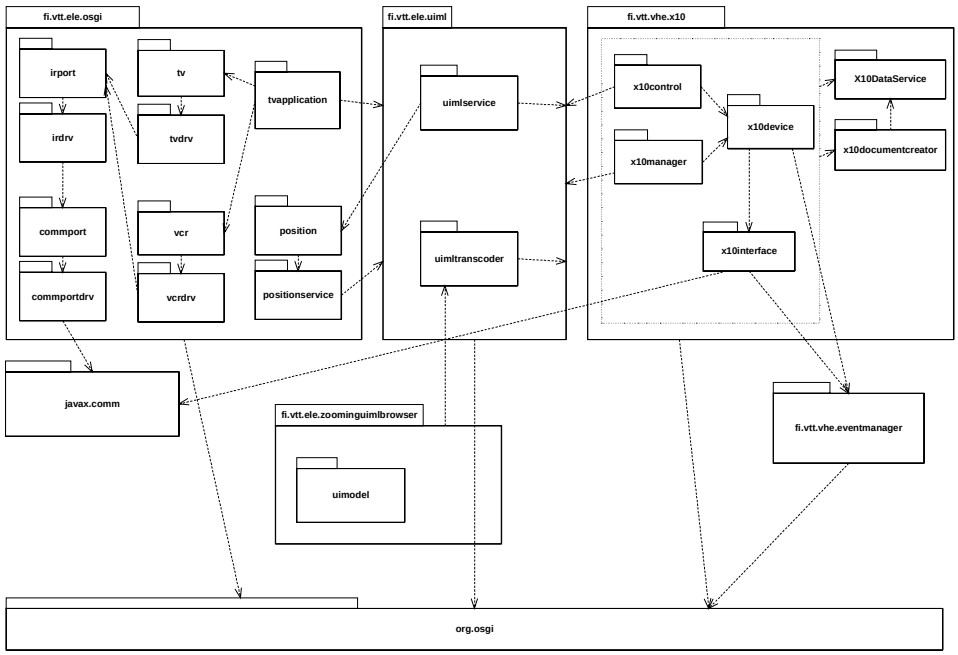

Figure 9. VHE-demonstrator’s package diagram. Illustration: Heikki Keränen.

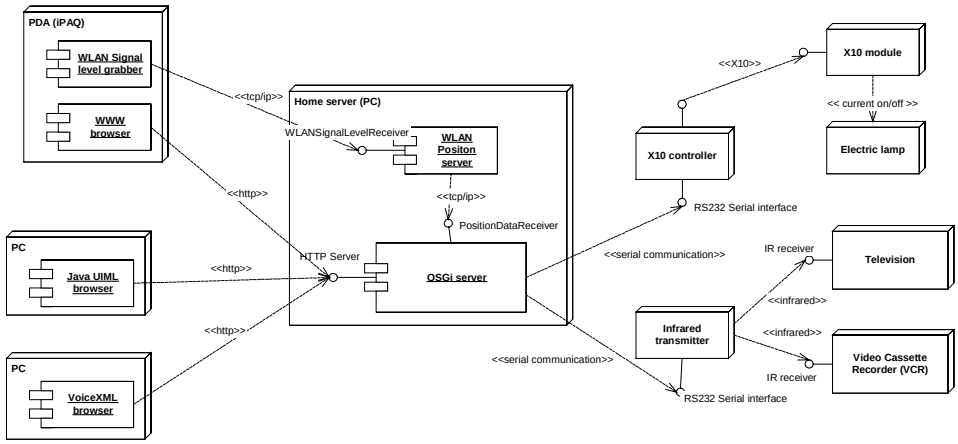

Figure 10. VHE-demonstrator's deployment diagram. Illustration: Heikki Keränen.

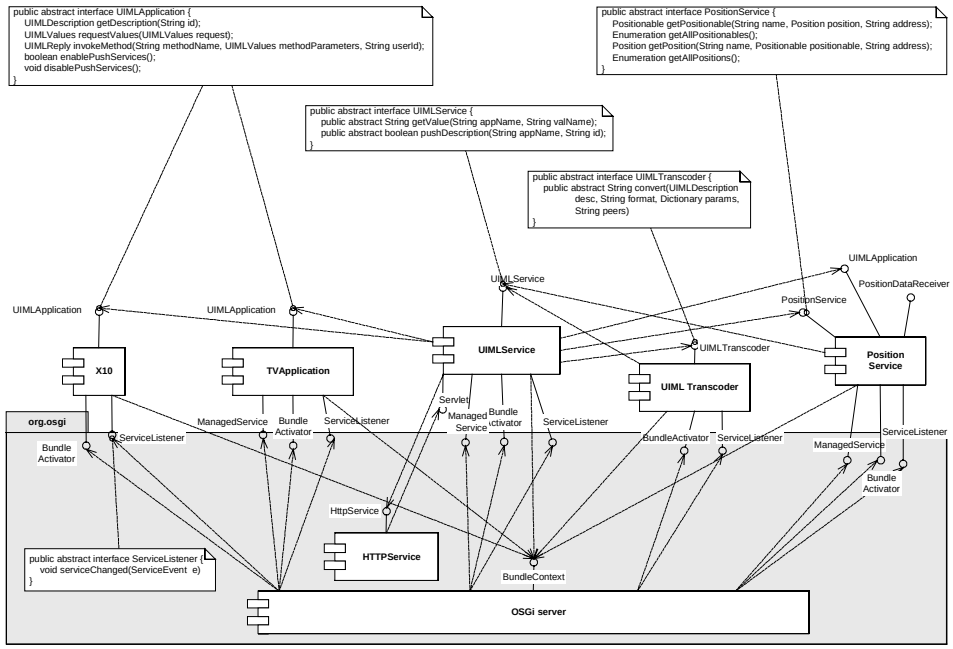

Figure 11. VHE demonstrator's OSGi-server component diagram. Illustration: Heikki Keränen.

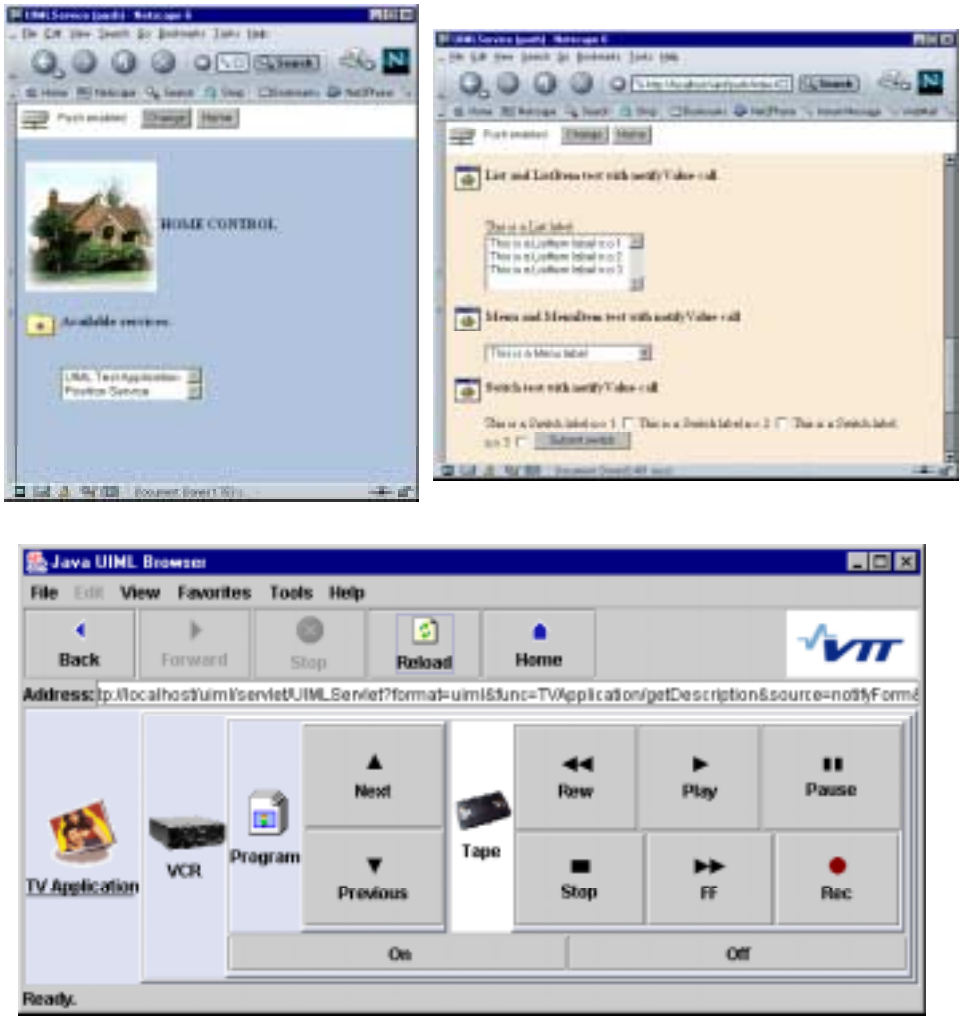

Figure 12. Screen captures of the UIML UIs.

3. References

1. UIML.org. *UIML Specification*. August 2002. URL: http:// www.uiml.org/ index.php

2. OSGi Consortium. *Open Service Gateway initiative*. August 2002. URL: http:// www.osgi.org

Appendix C: Abstract Requirements Specification of VHE

Owner: VTT Electronics

Scope: VHE

Originator: Eila Niemelä

Status: Accepted by PCC

Change History

Issue	*Date*	*Handled by*	*Comments*
0.1 Draft	18 Sep 2000	Eila Niemelä	Document is created.
0.2 Draft	26 June 2000	Hannu Rytilä	Classification modified and requirements added.
1.0 Final	12 Aug 2002	Markus Moilanen	Classification and requirements completed.

1. Purpose

The purpose of this specification is to describe the abstract requirements of the VHE system. The specification describes how these requirements should be mapped to the capabilities, qualities, and constraints of the Middleware used in different kinds of VHEs.

2. System Requirements

Virtual Home Environment in the sense of the ITEA/VHE project is a networked platform for home appliances allowing “Plug & Play” style communication and shared control between the in-home appliances, external services and mobile and

stationary terminals in order to support user-oriented services through a variety of interfaces.

VHE-Middleware is a set of services that enables a consumer to reach the VHE services across network and technology boundaries. The Middleware technology used in the information systems and the ongoing Middleware work done in the automation systems have encouraged researchers to develop analogous technologies for cost-critical systems that are targeted to the segmented mass markets.

2.1 Challenges

Here we will list some of the challenges seen in developing a VHE system.

2.1.1 Experience needed

- Knowledge of domain; reference software systems
- The incorporation of new technology/technologies
- The personnel on the architecture team

2.1.2 Legacy systems

- Why they are not valid any more?
- Is this a part of some larger development effort?
- Earlier patents?

2.1.3 Business context

- Market segments, critical functionality, critical quality attribute, technical constraints, and business constraints.
- Stakeholders involved in the product line.

2.2 Functional Requirements

Functional requirements or “Capabilities” of a system can be expressed in two main groups: one group describes the functionality that a new system is expected to provide for its users, and the other group describes the functionality that is considered to be normal for that type of system. The former group is explicitly expressed by end users. The latter group covers the functionality the end users will not explicitly state - they are implicit requirements - because the end users assume that the new system will be at least as good as the existing systems are normally.

2.2.1 Use Cases

Successful development of the software systems depends on the quality of the requirements engineering process. Use cases and scenarios are promising vehicles for eliciting, specifying and validating requirements.

2.2.2 Abstract Functional Requirements

Abstract functional requirements will be presented in two categories: the requirements which should be considered when designing the systems on the VHE Middleware, and the requirements which should be considered when designing the systems on the VHE Middleware itself. In this case, considering the first category, the UI-related approach will be presented. See Table 1 and Table 2.

Table 1. Abstract functional requirements for the UI of the VHE.

Classification	*Descriptions*
Remote and local access	Support for local and remote access to home services. Support for multiple access modes and devices such as wireless access through PDA or mobile phones in wireless or wired LAN. Support for integration of multiple networks "communities".
Smart integration	VHE will support plug and play integration between devices.
Manual integration	VHE will support manual integration of with non-VHE-compliant devices
Serverless interoperability	VHE will support customisable smart integration between any VHE devices without intervention of specific server device.
	VHE will support customisable smart integration between any VHE devices without intervention of specific server device.
Maintenance functions	Add and remove legacy devices and device drivers. Support an interface for customisation of smart integration rules. Configure the VHE for various feature sets. Network configuration.
Home owner's user interface	Check configuration including the features supported, devices attached, warranty dates for the VHE and each device, etc.
Diagnosis	Activate VHE diagnosis. Activate diagnosis for selected devices.
Modes	Night mode (not related to Middleware). Security mode. Diagnosis mode. Activate modes manually or automatically.
Basic device support	Turn on/off. Receive status. Reset. Start self-testing.

Table 2. Abstract functional requirements for the VHE Middleware.

Classification	*Descriptions*
Configuration management	Physical resources (devices, terminals, networks, servers etc.) Application services (directory service, naming service). Features/services of the middleware (properties of communication, configuration and security management)
Co-ordination management	Data-based. Control-based. Rule-based.
Co-operation management	Negotiation-based (agents). Message-based (asynchronous). Data-intensive (throughput). Command/control-based (reactivity). Mobility (mobile dynamic functionality)
Communication	Secure access of the home environment from the Internet. Internet accesses from home. Heterogeneity protocols (synchronisation, data format, routing, etc.) Local/distributed communication.
Security management	Maintains the integrity of objects and integrity of transactions between objects
Resource management	Management and sharing of persistent hardware resources. Management of changing SW resources Management of environmental and networked resources

2.3 Quality Requirements

The common classification of quality attributes will be introduced next.

2.3.1 PLA-related Quality Attributes

- **Modifiability** - the system's ability to make changes quickly and cost effectively (classes: extensibility, deleting unwanted capabilities, portability, restructuring)

- **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.

- **Reusability** - designing a system so that the system's structure or its components can be reused in future applications. (considered partly by integrability and modifiability).

- **Integrability** – the ability to make separately developed components of the system work together correctly.

- **Interoperability** - the ability of a group of parts (constituting a system) to work with another system.

- **Scalability** - the ease with which a system or component can be modified to fit the problem area.

- **Portability** – the ability of a system to run under different computing systems.

- **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.

2.3.2 Domain-related Quality Attributes

- **Performance** - the responsiveness of the system - the time required to respond to stimuli or the number of events processed in some interval of time.

- **Availability** - the proportion of time the system is up and running.

    - **Reliability** - the ability of the system or component to keep operating over time, or to perform its required functions under the stated conditions for a specified period.

    - **Survivability** - the ability to perform the designed set of functions, given software infrastructure component failures resulting in a service outage described by the number of services affected, the number of clients affected, and the duration of the outage.

- **Security** - a measure of the system's ability to resist unauthorised attempts at usage and denial of service while still providing its service to legitimate users.

- **Safety** - the needs of users to be protected against potential problems such as hardware or software faults.

- **Testability** - the ease with which the software can be made to demonstrate its faults testing.

- **Usability** - ease and comfort of using the system

    - **Learnability** - How quick and easy is for a user to learn to use the system's interface?

    - **Efficiency** - Does the system respond with appropriate speed to a user's requests?

    - **Memorability** - Can the user remember how to do system operations between uses of the system?

- **Error avoidance** - Does the system anticipate and prevent common user errors?

- **Error handling** - Does the system help the user recover from errors?

- **Satisfaction** - Does the system make the user's job easy?

2.4 Mapping the Quality-attributes for the VHE

Table 3 describes how a set of qualities should be mapped in the VHE system. Possible architectural mechanisms that might be used to realise a set of these qualities are presented in Table 4

.

Table 3. Abstract quality requirements of VHE.

Quality	*Mapping*
Modifiability	Device-related modifications: • add/remove known device types • Home owner modifications vs. Installer modifications • dynamic vs. static modifications Portability-related modifications: • move to new OS • move to new central processing unit Network-related modifications: • change from a wire-based to a wireless network • use of power lines for local area network • customise the smart interactions
Integrability	as described in Section 2.2.2
Performance	as described in Section 2.2.2
Availability	as described in Section 2.2.2
Reliability	as described in Section 2.2.2
Security	as described in Section 2.2.2
Testability	as described in Section 2.2.2
Usability	• For installer • For home owner • For children • Acknowledgement for commands • Independent verification Various input/output modes (speech, text)

Table 4. Options for architectural mechanisms.

Quality	*Mechanisms*
Modifiability	• Virtual interface for various device types • Layering for portability • Rule-based approach for customising smart interactions
Integrability	• Device interface standards • Message format standards • System-wide quality attribute models
Performance	Support for priority-based communications
Availablity (Reliability)	Redundancy between the VHE and devices when reliability is key
Security	• Firewall – single point of entry to the VHE • Encryption • Authentication – digital signature
Usability	• Support for undo, cancellation, etc. • No side effects

2.5 Constrains

Used standards and conformance to legacy systems.

Published by

Vuorimiehentie 5, P.O.Box 2000, FIN–02044 VTT, Finland
Phone internat. +358 9 4561
Fax +358 9 456 4374

Series title, number and report code of publication

VTT Publications 476
VTT–PUBS–476

Author
Moilanen, Markus

Title

Middleware for Virtual Home Environments
Approaching the Architecture

Abstract

Virtual Home Environments (VHE) is the concept that networks supporting mobile users should provide them with the same computing environment on the road as they are used to having in their home or corporate computing environment. The VHE Middleware project is one of the ITEA projects. The goal of the project is to make European industry the leader in Middleware software technology for end-user terminals, with wireless connections and the corresponding infrastructure to enable VHE. This publication documents the R&D of VHE Middleware carried out by VTT Electronics. A full case study evaluating an advanced method of identifying the conceptual architecture of a SW system from its functional requirements is presented. The proposed method is based on the common knowledge of object-oriented analysis (OOA) methodology, which claims that every software system contains hierarchical structures that reflect its functional requirements. In OOA methodology, these structures and their hierarchies can be found by analysing and structuring the problem descriptions - the Use Case analysis. The advanced method here is how to seamlessly move from a use case model to a conceptual architecture model of the software system. User's scenarios and software prototypes of the VHE system are used as a case study. Using the proposed method, the VHE system's subsystems, layers and APIs have been found and the conceptual architecture of VHE Middleware has been drawn up. After this, the technical development of the case is extended to fully concrete the VHE Middleware architecture and its elements.

Keywords

software systems, software layers and subsystems, conceptual architecture, object-oriented analysis OOA, application programming interfaces APIs, unified modelling language UML, user interface mark-up language UIML, generic user interface, remote service development, serverless service provisioning

Activity unit

VTT Electronics, Kaitoväylä 1, P.O.Box 1100, FIN-90571 OULU, Finland

ISBN			Project number
951–38– 6003-5 (soft back ed.) 951–38–6004-3 (URL:http://www.inf.vtt.fi/pdf/)			
Date	Language	Pages	Price
November 2002	English	115 p. + app. 46 p.	D
Name of project		Commissioned by	
Series title and ISSN VTT Publications 1235–0621 (soft back ed.) 1455–0849 (URL: http://www.inf.vtt.fi/pdf/)		Sold by VTT Information Service P.O.Box 2000, FIN–02044 VTT, Finland Phone internat. +358 9 456 4404 Fax +358 9 456 4374	

VTT PUBLICATIONS

458 Karema, Hannu. Numerical treatment of inter-phase coupling and phasic pressures in multi-fluid modelling. 2002. 62 p. + app. 51 p.

459 Hakkarainen, Tuula. Studies on fire safety assessment of construction products. 2002. 109 p. + app. 172 p.

460 Shamekh, Salem Sassi. Effects of lipids, heating and enzymatic treatment on starches. 2002. 44 p. + app. 33 p.

461 Pyykönen, Jouni. Computational simulation of aerosol behaviour. 2002. 68 p. + app. 154 p.

462 Suutarinen, Marjaana. Effects of prefreezing treatments on the structure of strawberries and jams. 2002. 97 p. + app. 100 p.

463 Tanayama, Tanja. Empirical analysis of processes underlying various technological innovations. 2002. 115 p. + app. 8 p.

464 Kolari, Juha, Laakko, Timo, Kaasinen, Eija, Aaltonen, Matti, Hiltunen, Tapio, Kasesniemi, Eija-Liisa, & Kulju, Minna. Net in Pocket? Personal mobile access to web services. 2002. 135 p. + app. 6 p.

465 Kohti oppivaa ja kehittyvää toimittajaverkostoa. Tapio Koivisto & Markku Mikkola (toim.). 2002. 230 s.

466 Vasara, Tuija. Functional analysis of the RHOIII and 14-3-3 proteins of *Trichoderma reesei.* 93 p. + app. 54 p.

467 Tala, Tuomas. Transport Barrier and Current Profile Studies on the JET Tokamak. 2002. 71 p. + app. 95 p.

468 Sneck, Timo. Hypoteeseista ja skenaarioista kohti yhteiskäyttäjien ennakoivia ohjantajärjestelmiä. Ennakointityön toiminnallinen hyödyntäminen. 2002. 259 s. + liitt. 28 s.

469 Sulankivi, Kristiina, Lakka, Antti & Luedke, Mary. Projektin hallinta sähköisen tiedonsiirron ympäristössä. 2002. 162 s. + liiitt. 1 s.

471 Tuomaala, Pekka. Implementation and evaluation of air flow and heat transfer routines for building simulation tools. 2002. 45 p. + app. 52 p.

472 Kinnunen, Petri. Electrochemical characterisation and modelling of passive films on Ni- and Fe-based alloys. 2002. 71 p. + app. 122 p

473 Myllärinen, Päivi. Starches – from granules to novel applications. 2002. 63 p. + app. 60 p.

474 Taskinen, Tapani. Measuring change management in manufacturing process. A measurement method for simulation-game-based process development. 254 p. + app. 29 p.

475 Koivu, Tapio. Toimintamalli rakennusprosessin parantamiseksi. 2002. 174 s. + liitt. 32 s.

476 Moilanen, Markus. Middleware for Virtual Home Environments. Approaching the Architecture. 2002. 115 p. + app. 46 p.

477 Purhonen, Anu. Quality driven multimode DSP software architecture development. 2002. 150 p.

478 Abrahamsson, Pekka, Salo, Outi, Ronkainen, Jussi & Warsta, Juhani. Agile software development methods. Review and analysis. 2002. 107 p.

479 Karhela, Tommi. A Software Architecture for Configuration and Usage of Process Simulation Models. Software Component Technology and XML-based Approach. 2002. 129 p. + app. 19 p.

480 Laitehygienia elintarviketeollisuudessa. Hygieniaongelmien ja *Listeria monocytogeneksen* hallintakeinot. Gun Wirtanen (toim.). 2002. 183 s.

481 Wirtanen, Gun, Langsrud, Solveig, Salo, Satu, Olofson, Ulla, Alnås, Harriet, Neuman, Monika, Homleid, Jens Petter & Mattila-Sandholm, Tiina. Evaluation of sanitation procedures for use in dairies. 2002. 96 p. + app. 43 p.

482 Wirtanen, Gun, Pahkala, Satu, Miettinen, Hanna, Enbom, Seppo & Vanne, Liisa. Clean air solutions in food processing. 2002. 93 p.

483 Heikinheimo, Lea. *Trichoderma reesei* cellulases in processing of cotton. 2002. 77 p. + app. 37 p.

484 Taulavuori, Anne. Component documentation in the context of software product lines. 2002. 111 p. + app. 3 p.

Virtual Home Environments (VHE) is the concept that networks supporting mobile users should provide them the same computing environment on the road as they are used to having in their home or corporate computing environment. The VHE Middleware project is one of the ITEA projects. The goal of the project is to make European industry the leader in Middleware software technology for end-user terminals, with wireless connections and the corresponding infrastructure to enable VHE. This publication documents the R&D on VHE Middleware carried out by VTT Electronics. A full case study evaluating an advanced method of identifying the conceptual architecture of a SW system from its functional requirements is presented. The proposed method is based on the common knowledge of object-oriented analysis (OOA) methodology, which claims that every software system contains hierarchical structures that reflect its functional requirements. In OOA methodology, these structures and their hierarchies can be found by analysing and structuring the problem descriptions - the Use Case analysis. The advanced method here is how to seamlessly move from a use case model to a conceptual architecture model of the software system. User's scenarios and software prototypes of the VHE system are used as a case study. Using the proposed method, the VHE system's subsystems, layers and APIs have been found and the conceptual architecture of VHE Middleware has been drawn up. After this, the technical development of the case is extended concretising the VHE Middleware architecture and its elements.

Tätä julkaisua myy
VTT TIETOPALVELU
PL 2000
02044 VTT
Puh. (09) 456 4404
Faksi (09) 456 4374

Denna publikation säljs av
VTT INFORMATIONSTJÄNST
PB 2000
02044 VTT
Tel. (09) 456 4404
Fax (09) 456 4374

This publication is available from
VTT INFORMATION SERVICE
P.O.Box 2000
FIN–02044 VTT, Finland
Phone internat. +358 9 456 4404
Fax +358 9 456 4374

ISBN 951–38–6003–5 (soft back ed.)
ISSN 1235–0621 (soft back ed.)

ISBN 951–38–6004–3 (URL: http://www.inf.vtt.fi/pdf/)
ISSN 1455–0849 (URL: http://www.inf.vtt.fi/pdf/)

VTT PUBLICATIONS 476 Middleware for Virtual Home Environments. Approaching the Architecture Markus Moilanen