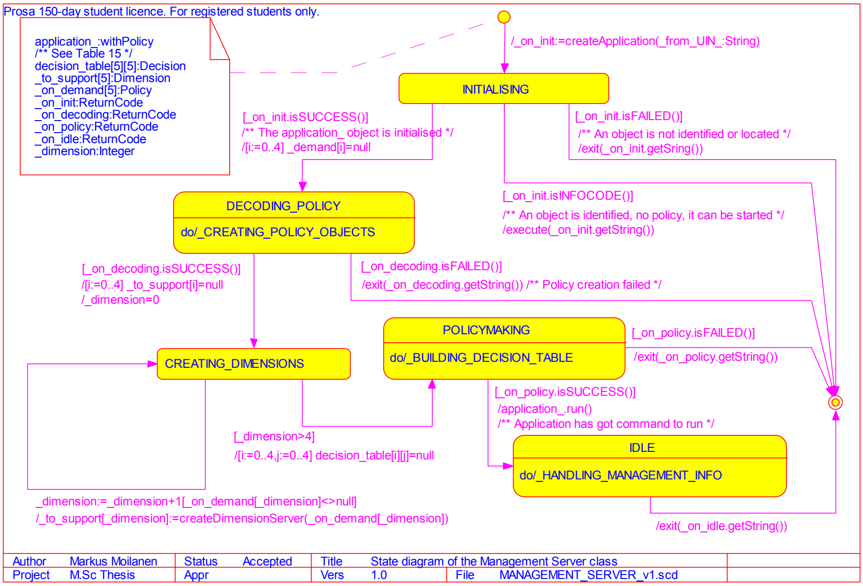
442 VTT PUBLICATIONS

Markus Moilanen

Management framework of distributed software objects and components


Prosa 150-day student licence. For registered students only.
application_:withPolicy
/** See Table 15 */
decision_table[5][5]:Decision
_to_support[5]:Dimension
_on_demand[5]:Policy
_on_init:ReturnCode
_on_decoding:ReturnCode
_on_policy:ReturnCode
_on_idle:ReturnCode
_dimension:Integer
/_on_init:=createApplication(_from_UIN_:String)
INITIALISING
[_on_init.isSUCCESS()]
/** The application_ object is initialised */
/[i:=0..4] _demand[i]=null
[_on_init.isFAILED()]
/** An object is not identified or located */
/exit(_on_init.getSring())
DECODING_POLICY
do/_CREATING_POLICY_OBJECTS
[_on_init.isINFOCODE()]
/** An object is identified, no policy, it can be started */
/execute(_on_init.getString())
[_on_decoding.isSUCCESS()]
/[i:=0..4] _to_support[i]=null
/_dimension=0
[_on_decoding.isFAILED()]
/exit(_on_decoding.getString()) /** Policy creation failed */
POLICYMAKING
do/_BUILDING_DECISION_TABLE
[_on_policy.isFAILED()]
/exit(_on_policy.getString())
CREATING_DIMENSIONS
[_on_policy.isSUCCESS()]
/application_.run()
/** Application has got command to run */
[_dimension>4]
/[i:=0..4,j:=0..4] decision_table[i][j]=null
IDLE
do/_HANDLING_MANAGEMENT_INFO
_dimension:=_dimension+1[_on_demand[_dimension]<>null]
/_to_support[_dimension]:=createDimensionServer(_on_demand[_dimension])
/exit(_on_idle.getString())
Author Markus Moilanen
Status Accepted
Title State diagram of the Management Server class
Project M.Sc Thesis
Appr
Vers 1.0
File MANAGEMENT_SERVER_v1.scd

TECHNICAL RESEARCH CENTRE OF FINLAND ESPOO 2001

VTT PUBLICATIONS 442

Management framework of distributed software objects and components

Markus Moilanen

VTT Electronics

TECHNICAL RESEARCH CENTRE OF FINLAND
ESPOO 2001

ISBN 951–38–5863–4 (soft back ed.)
ISSN 1235–0621 (soft back ed.)

ISBN 951–38–5864–2 (URL:http://www.inf.vtt.fi/pdf/)
ISSN 1455–0857 (URL:http://www.inf.vtt.fi/pdf/)

JULKAISIJA – UTGIVARE – PUBLISHER

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

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

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

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

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

VTT Electronics, Embedded Software, 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 2001

Moilanen, Markus. Management framework of distributed software objects and components. Espoo 2001. Technical Research Centre of Finland, VTT Publications 442. 153 p. + app. 45 p.

Keywords software processes, software design, quality methods, requirement specifications, conceptual planning, application layer, management layer, component architecture, UML

Abstract

The deployment of software applications without making any arrangements beforehand in the target platform sets strict requirements for the management of the applications and the management of the platform in which the application will be executed. In this work, "objective quality thinking" concerning the work process has been chosen to gain a solution for this problematic issue. In the work, the management framework of the distributed applications is developed. Application software is understood to be any useful piece of software that can be provided.

The developed management framework observes different dimensions of the management. Dimension is such a feature group of the management that can be implemented independently of other feature groups. These dimensions include for example the advertisement of a component, distribution of a component, resource management of a component, and security of a component. The dimensions are presented starting from the conceptual level ending to the explicit design models. These design models can be implemented for any application-specific purpose by implementing the presented application programming interfaces of the models. Lastly in the work, an example of how to apply the management to an application is presented. The case considers an application which distributes itself according to its associated distribution policy, and which is implemented for the Internet environment in the Java programming language.

The presented solution of the management can be applied to any data objects, including passive objects. If extended to its full design, the solution would be a starting point for a new software management technology.

Preface

The author has been studying the subject of this thesis during several projects at the Technical Research Centre of Finland, department of Electronics (VTT Electronics), since 1st June 1999. One of the projects, called DYRE, dealt with the issue of "dynamic resource management". Another project, called EMU, and particularly its sub-project, called LONTONEXTG, was named to "develop some gateway-technology to assist the distribution of home-services". The subject is also involved in the European-wide ITEA-project (http://www.itea-office.org/), which is developing the technique for the virtual home environment. The main result of these projects - considering the work of the author - was an experimental distribution concept started to be developed for testing purposes. The work is now continuing as a M.Sc. thesis and this makes it possible for the author to finish the work without any restrictions from the formal project, and with increasing creativity. The thesis is carried out under the organisation of VTT Electronics (http://www.ele.vtt.fi). Personal contacts to the author can be made by e-mail at mmoi@iki.fi.

I would like to thank Professor Tino Pyssysalo, who has been the supervisor of this thesis, and Professor Juha Röning, the second examiner of this thesis, at the University of Oulu. Prof. Pyssysalo supported me with valuable comments, guidance, demands, and encouragement during this duty. I would also like to thank Prof. Veikko Seppänen for his most valuable expert's survey report concerning the document proposal before the final publication of the work.

Furthermore, I would like to express my deepest gratitude to Mr. Hannu Rytilä, who represented the employer of this thesis, and Mr. Hannu Honka, from VTT Electronics. They made my work possible by providing the working place, the topical subject and time to study it, and the valuable advice, even when the result of the work was unseen. Ms. Marjo Jussila, also from VTT Electronics, deserves my gratitude because of her work of correcting my writing of English.

Finally, to all who may feel that they have contributed to the work - thank you for the support.

Oulu, Finland, 30 May 2001 Markus Moilanen

Contents

Appendices

List of symbols

ANSA	Advanced Network Systems Architecture
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ASN.1	Abstract Syntax Notation, version 1
BAA	Business Area Analysis
CCM	CORBA Component Model
CLD	Class Diagram
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
CSCW	Computer Supported Co-operative Work(ing)
CSMA/CD	Carrier-Sense Multiple Access with Collision Detection
COTS	Commercial Off-The-Shelf component
DCE	Distributed Computing Environment
DCOM	Distributed Component Object Model
DDE	Dynamic Data Exchange
DNS	Domain Name Service
DOS	Disk Operating System

EJB	Enterprise JavaBeans
ETSI	European Telecoms Standards Institute
FTP	File Transfer Protocol
GPRS	General Packet Radio Service
GSM	Global System for Mobile Communications
HLDC	High-Level Datalink Control
HTTP	Hyper-Text Transfer Protocol
ICA	Independent Computing Architecture
IEC	International Electrotechnic Commission
IIOP	Internet Inter-ORB Protocol
IP	Internet Protocol
IPX/SPX	Inter-network Packet Exchange protocol/Sequence Packet Exchange - Novell protocol family for internetworking
ISDN	Integrated Services Digital Network
ISO	International Standards Organisation
ITU	International Telecommunication Union
ITU-T	The ITU Telecommunication Standardisation Sector (ITU-T) is one of the three Sectors of the International Telecommunication Union (ITU).
IXIT	Implementation Extra Information for Testing

JDK	Java Development Kit
JVM	Java Virtual Machine
LDAP	Lightweight Directory Access Protocol
MExE	Mobile Station Application Execution Environment
MSD	Message Sequence Diagram
MTS	Microsoft Transaction Server
NetBEUI	NetBIOS Extended User Interface
NetBIOS	Network Basic Input Output System
OCX	OLE custom controls
OLE	Object Linking and Embedding
OMG	Object Management Group
OO(A)	Object-Oriented (Analysis)
ORB	Object Request Broker
OS	Operating System
OSF	Open Software Foundation
OSGi	Open Services Gateway initiative
OSI	Open Systems Interconnection
PC	Personal Computer
PDA	Personal Digital Assistant

PDL	Program Design Language
PLA	Product Line Architecture
QFD	Quality Function Deployment
OTS	Off-The-Shelf component
QoS	Quality of Service
R&D	Research and Development
RDP	Remote Desktop Protocol
RMI	Remote Methods Invocation
RM-ODP	Reference Model for Open Distributed Processing
ROBO	Remote Office/Branch Office
RPC	Remote Procedure Calling
SCD	State Chart Diagram
SDK	Software Developers Kit or Software Development Kit
SCM	Software Configuration Management
SMG4	ETSI's GSM standards setting body, standardisation group for GSM data
SMS	Short Message Service
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol

SOHO	Small Office/Home Office
T.120	ISO/ITU-T that describes a data protocols for multimedia conferencing
TCP/IP	Transmission Control Protocol/IP
TINA-C	Telecommunication Information Networking Architecture - Consortium
UID	Unique IDentifier
UIN	UnIque Name
UML	Unified Modelling Language
UMTS	Universal Mobile Telecommunications System
URI	Universal Resource Identifier
URL	Uniform Resource Locator, specialisation of the URI
WAP	Wireless Application Protocol
WWW	World Wide Web
X.21	The standard of ISO/ITU-T that describes a synchronous communication at physical level
X.25	The standard of ISO/ITU-T that describes a packet transfer protocol with virtual circuits
X.400	The standard of ISO/ITU-T that describes a mail service
X.500	The standard of ISO/ITU-T that describes a directory service
XML	Extended Mark-up Language

1. Introduction

The assignment state of this work is to research and develop some concepts of techniques to manage a piece of the application software to be executed on distributed, maybe mobilised, software execution environment. The management problem on a large scale exists because of the heterogeneous, maybe mobile, executing platform and because of the users of today and in the near future. The objective is to release users from solving the problems of technical nature, as is the case today even with simple PC (Personal Computer).

Application software is understood to be any piece of software that can be provided, from deploying one byte to deploying a mail service - large systems of that kind are usually provided by deploying some component model.

As a result, there should be a framework for managing distributed applications in their environment. The framework can be used when the research of distributed software continues further concentrating to examine the fields of problems that will exist with the upcoming wireless and mobile communication technology. Furthermore, the result should contain the definitions of the APIs (Application Programming Interfaces) - this is supposed to help in exploiting the result in existing concepts of distributed application and operational systems as an increment and complement.

A quality focus and the use of customer-oriented methods must be set as a basis for the work

1.1 Approach

An end-user of the information technology lives in a world where he is required to be almost an engineer to understand and deploy the different equipment and related software surrounding the end-user. Even more, devices mentioned to be used by the end-user are increasing rapidly so that before long, the end-user is using some equipment or software at every turn. However, as a paradox, technology itself seems to be engineered only for the technician. The described situation has been led into by technicians, people who think about systems from their own professional perspective, viewing the systems from inside. They have made the end-user to carry the load and pay for its costs without asking how the end-user feels.

In this work, the customer-oriented way of thinking has been chosen to achieve a solution to the paradox. Firstly, we will agree with Bill Joy, one of the creators and champions of Jini, who has stated, "Large successful systems start out as small successful systems" [1]. This philosophy is part of the Jini motivation. Secondly, the statement has been developed further here by promoting the next: "If we will find out the requirements for the largest system, we will know which the requirements for the smallest system might be - then we will simply develop the management model for any piece of the software by deploying the requirements found". Thirdly, the proper tools and methods for support will be chosen to thoroughly help find out the user requirements, and furthermore, proper tools and methods will be chosen to maintain the demands of the requirements from beginning to end.

A quality focus is a very important factor in product development today [2, 3]. Working under "ISO9001-sertified" quality system (as is the case here) is not enough; quality must be built into a product by using appropriate tools, methods and processes [2]. The quality of the software product is twofold:

1. The fitness for use and the ability to meet the customer’s expectations.

2. Meeting Specifications and Requirements.

The first is something that can be called as objective quality, something that is reached for. The second is something that appears in the real product, something can be called as subjective quality, i.e. quality that must be built into the product by a professional with respect for rules to support different subjective quality properties of the software product. There are many categories of those properties developed during the times, for example McCall's quality factors [4], or "ISO 9126 Quality Factors". Those factors will appear and will be mentioned at an appropriate point as this process progresses.

1.2 Research methods

Figure 1 illustrates how the author sees the position of an individual and organisation working on software production, as follows:

- Attempts will be made to satisfy customer needs that come from the market.

- However, only some of the possible offers are accepted by the market, only few of them will continue to live.

- Are we working mostly for nothing?

- The answer is no because we can increase our competitiveness and productivity using an advanced software process.

- Both get richer - the individual increases his tacit knowledge and organisation increases its resources.

The situation with a scientific study in the area of software research is much the same with the one of software production.

Iivari [5] defines three major categories of the research methods of information systems: constructive (including conceptual and technical development), nomothetic (including formal-mathematics analysis, experiments and field studies), and idiographic (including cases and action research) ones. Iivari's constructive research method can be compared to a software process because the method supports a lifecycle of an artifact like software processes consistently do. Järvinen et al. [6] categorise the research methods in the five major groups. One of the suggested group, the constructive research, match on software processes by its specification to implementation sequence.

The author suggests a process model in which the conceptual-analytical research method is applied before the use of the constructive research method.

The Road Map depicted in Figure 2 gives ideas put in together. It is for understanding the quality and productivity based working process used to solve the research problem here.

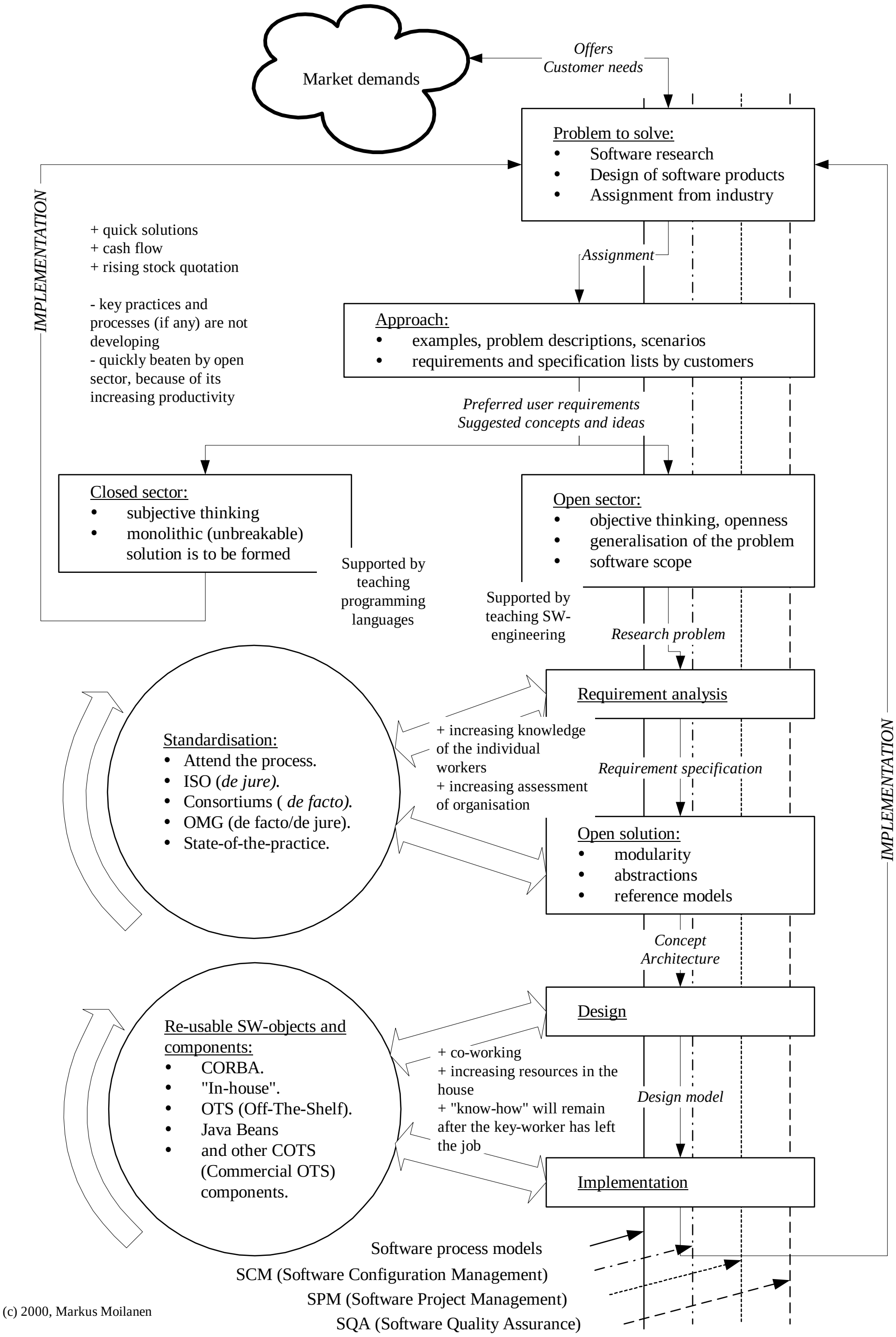

Market demands
Offers
Customer needs
Problem to solve:
• Software research
• Design of software products
• Assignment from industry
IMPLEMENTATION
+ quick solutions
+ cash flow
+ rising stock quotation
- key practices and processes (if any) are not developing
- quickly beaten by open sector, because of its increasing productivity
Assignment
Approach:
• examples, problem descriptions, scenarios
• requirements and specification lists by customers
Preferred user requirements
Suggested concepts and ideas
Closed sector:
• subjective thinking
• monolithic (unbreakable) solution is to be formed
Supported by teaching programming languages
Open sector:
• objective thinking, openness
• generalisation of the problem
• software scope
Supported by teaching SW-engineering
Research problem
Requirement analysis
Standardisation:
• Attend the process.
• ISO (de jure).
• Consortiums (de facto).
• OMG (de facto/de jure).
• State-of-the-practice.
+ increasing knowledge of the individual workers
+ increasing assessment of organisation
Requirement specification
IMPLEMENTATION
Open solution:
• modularity
• abstractions
• reference models
Concept
Architecture
Design
Re-usable SW-objects and components:
• CORBA.
• "In-house".
• OTS (Off-The-Shelf).
• Java Beans
• and other COTS (Commercial OTS) components.
+ co-working
+ increasing resources in the house
+ "know-how" will remain after the key-worker has left the job
Design model
Implementation
Software process models
SCM (Software Configuration Management)
SPM (Software Project Management)
SQA (Software Quality Assurance)
(c) 2000, Markus Moilanen

Figure 1. Processes of software production.

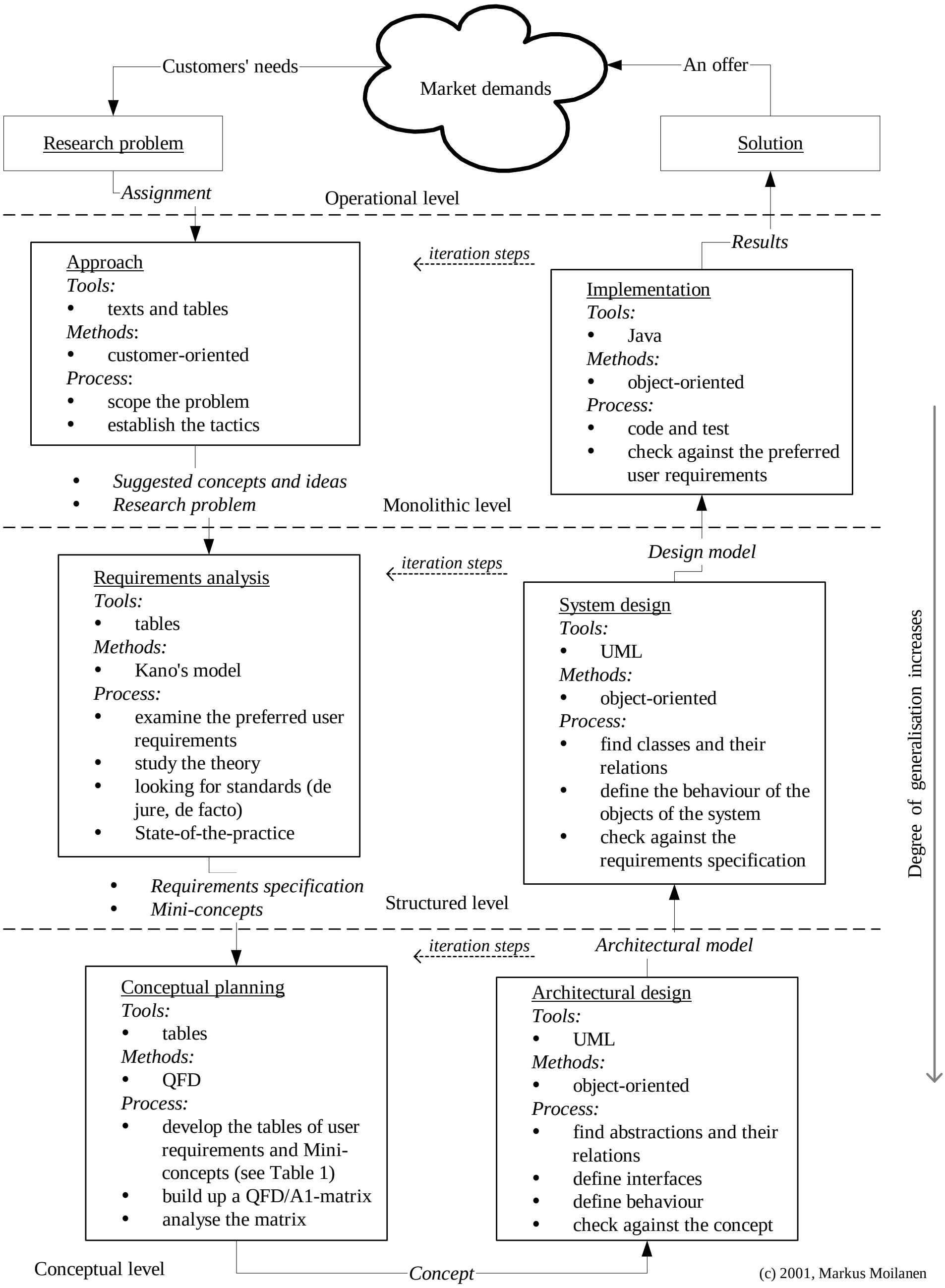


Figure 2. The Road Map.

The map first shows how the problem is generalised until the conceptual level is reached and then how to build the solution upon the concept. The V-model test pattern of software is imperceptible, too. Figure 2 also shows how iteration steps can be applied over one level or even over many levels during R&D-phases. For example, if working on the monolithic level, solutions can be checked only against the user requirements of this level, namely the preferred user requirements, but the necessary iteration steps can be extended to all levels under the monolithic one.

Further, the conceptual level is the border between changing from asking “what should we do” to answering to “how will we do it”. Again, with “what” the objective quality is applied and with “how” the subjective quality correspondingly.

Figure 2 also shows how the tools, methods and processes are used for this work particularly:

- Tools to be used are textual and graphical tools for presentations, tables and matrix-tools to organise knowledge and UML (Unified Modelling Language) to model the solutions.

- Methods to be used are Kano Model to ascertain that all customer (user) requirements are taken into consideration, and QFD (Quality Function Deployment) [3, 7] to make sure that the requirements are also included in the solution. The OO (Object-Oriented) method is used in all models and programming.

- Processes express how the work flows and what are inputs and outputs during the work progress. Overall the process model used in R&D (Research and Development) is a spiral model [2], and it makes flexible iteration steps possible.

When comparing the presented process framework to the one usually used with UML, it appears that the Use-case analysis is not used. Even if the Use-case analysis is a powerful tool with many, even abstract, problems, it does not fill the role of easy-to-understand way to present our analysis of the purpose we have. However, after the conceptual level has been reached, Use-case analysis tools will be used to present the analysis of the more specific technical requirements to be realised. The use of extended QFD-matrix is also possible -

choices will depend on the issue of whom the presentation is intended for, for technicians or users.

1.3 Distributed software objects and components

Software objects that are based on different component models are building parts of software applications. Objects might be distributed over a computer network.

According to Niemelä [8], the component framework of a distributed system introduces two dimensions: tiers and elements. The three tiers of the component framework define the subsystem in the first tier, integration platform in the second tier, and the product family in the third tier. The tiers explain the domain, technology, and business viewpoints of the frameworks, correspondingly. The elements define the product features, software architecture, components and their interaction mechanisms. The component framework can be depicted by architecture styles, key-mechanisms, services and components of each tier. The development of the component framework must be presented by the development of the different reuse assets.

Another view of the component framework is presented by Gamma et al. [9]; the design patterns are the basis of the development of reusable component-based software. Design patterns describe simple and elegant solutions to specific problems in object oriented software design. Design patterns capture solutions that have developed and evolved over time.

A new term will be introduced in this thesis: a mini-concept. Mini-concepts are techniques in the existing distributed systems by which the user requirements have been demanded until this day. Mini-concepts express how those techniques perform the job in practice. A mini-concept can be derived from a design pattern, an architecture pattern, a key-mechanism, a design rule, or some other feature that can be found under the development of the practical systems. However, after the solution is formed, the patterns, mechanisms, and rules will appear again.

1.4 Assumptions

Solving problems is a creative process. Many ideas may come up and many may die during the process. However, some ideas may still succeed to defend their justification to live. In this section, ideas and visions originating from the pre-working phase are presented. The pre-work comprises 1) the author's work in the DYRE project to develop his idea of the decomposing of the distributed computing problem domain into the orthogonal group of services which commonly exist with the distributable objects, and 2) the author's work in the LONTONEXTG project [10] (in which the author worked as a project manager, system analyst, and a main programmer) of designing the software for the home service distribution platform. Some initial suggestions from the past projects are listed in Table 1. The author's work in the DYRE project was originally intended to be the pre-work phase for the forthcoming diploma thesis - this diploma thesis - and hereby the thesis documents the DYRE project. This work thus starts with the best ideas from the past projects, and then develops the ideas to become a mature solution of the management of the services of the distributable objects.

Table 1. Initial suggestions of concepts and ideas.

No.	Concept	Note
1.	Application program should be such that only the part of an application that is absolutely needed to present the application to the end-user, will be loaded onto a memory of the end-user terminal device. This loadable part of the application would work as a bridge between itself and its user, the application itself still being able to remain anywhere in the network. The application itself could be as large as necessary and be combined partially from some other services in the network. End-user should see only the part of the application that is necessary for using it, and the user may pay only by usage.	This concept is currently used at least by Jini, [1] and Microsoft.NET remoting [11] techniques. The *representative* of an application is called a *proxy*.
The table continues.		

Continuing Table 1.

No.	Concept	Note
2.	When loading it, it should be taken care of how to configure and arrange resources for the application to enable its working in the end-user terminal device. A multiple allocation of resources can be avoided when every individual service exists only once in some specific interior.	Resource management functions are almost unsupported today.
3.	The application itself may maintain its function group of security, advertisement, distribution and other maintenance according to a predefined strategy defined by the *policy*. • Can these function groups be pointed to be *orthogonal* compared to each another - concerning *dimensions* of software objects? • Can the existence of the dimensions of software objects be proved by means of *mini-concepts*?	Some of these features are presented in RM-ODP (Reference Model for Open Distributed Processing) by its function groups [12].
4.	Applications programmers should program only the application logic, not the support for distribution. • Can the consequences of unstable distributed environment really be masked out totally to offer programmers to become isolated from the difficulties of programming for distributed environment? • Can we assume that there should be a *layer* of application level and the layer of management level included in underlying systems?	RM-ODP's distribution transparencies [12].

1.5 Research problem

A framework to manage the distributed use of a piece of computer software has to be developed. It should be as follows:

- the user requirements must be examined,

- initial suggestions of concepts and ideas presented in Table 1 must be evaluated, and if the dimensions can be proved to exist, and - as a consequence - further management functions to be isolated, then the functions must be developed further, this thereby leading to the implementation of the system,

- the concept must be defined on the level of abstractions and further design models so it can be realised for many purposes, for instance as a test platform which is easy to understand and deploy,

- definitions for the APIs of the system must be defined, and

- eventually, a demo program for deploying the solution must be presented.

2. Requirements analysis

In this chapter, requirements analysis of the system will be carried out. Requirements tell us what we have to demand from the viewpoint of customers. When quality was defined in Section 1.1, it was found out that the quality of a software product is twofold, including 1) the ability to meet the users' expectations - objective quality was thereby discussed, 2) the ability to meet the specifications and requirements - the subjective quality was thereby presented. For the first item, user's expectations of the system will be considered, and for the second item, the specifications and requirements that may concern the distributed system will be clarified. The method to be used for the requirements analysis is the Kano Model. By this, the aim is to combine the objective and subjective quality requirements and thus provide the data for concept planning in Chapter 3.

The Kano Model measures the level of satisfaction with a product against the consumer perceptions of attribute performance. Kano argues that attributes can be classified into three categories. Definitions may vary by the source, but the meaning is the same. Definitions of the three categories mentioned are listed below [13]:

- Performance characteristics exhibit a linear relationship between perceptions of attribute performance and customer satisfaction. A strong performance on these 'need' attributes enhances, while a weak performance reduces, satisfaction with the product or service. Adding more attributes of this type to a product will also raise customer satisfaction. Examples of performance characteristics include the duration of rechargeable battery life of a cellular telephone.

- Excitement characteristics are unexpected attributes that, when provided, disproportionately generate high levels of customer enthusiasm and satisfaction. When these 'nice-to-have' attributes are not available in a product, it does not lead to customer dissatisfaction. Examples of excitement characteristics include a CD player included as a standard piece of equipment in an economy car.

- Threshold characteristics provide diminishing returns in terms of customer satisfaction. These are essential or 'must' attributes of performance and do not offer any real opportunity for product differentiation. Providing threshold attributes and meeting customer expectations will do little to enhance overall customer satisfaction, but removing them or performing poorly on them will damage customer satisfaction. Examples of threshold characteristics include timely delivery of a magazine subscription, and the ever-present telephone dial tone.

User needs and nice-to-have requirements will be presented in Section 2.1. To collect must-requirements, a study of theory, standards and comparable techniques must be carried out. A set of practical systems will be examined to find out what the requirements seen by the designers of every practical system are, and to find out mini-concepts related of the system. This will be done in Section 2.2. Mini-concepts will be presented in Section 2.3.

2.1 User's requirements as performance and excitement

In this section, BAA (Business Area Analysis) will be carried out. Consequently, stakeholders and their preferred requirements for the system to be developed will be found out. These requirements may be considered as the reason for developing this particular system. At the same time, we will find out the vocabulary being used with the comparable system that customers refer to. When the end of the job with implementation has been reached, the software can be validated against these preferred user requirements.

End-user

Today, from the viewpoint of the end-user of information technology - as an ordinary consumer - a usual PC with the Internet connection capability used in homes [14], and on the other hand, the movable, maybe pervasive, handheld or virtual reality terminal in eye-classes meant for everyone in the future, which exists in a visionary's mind or on the planning board, both actually demand the same basic need expressed by the end-user - to enable access to the communion with information society. The Internet-explosion at the beginning of the 90's brought many new possibilities for end-users to communicate with each other and to reach and use information services. The end-user could also start to provide contents in the net by publishing WWW-pages (World Wide Web), and to communicate globally on a personal level using e-mail. He was provided with a huge global storage of all kinds of software and information services, but, as a back impact, the security in using offered services was weakened dramatically.

Software

The concept of executable program mode (EXE) [15] was found to be weak against all kinds of phenomena which were threatening its integrity. Examples of these threats include viruses and unauthorised copy. Software viruses or some other malfunctions or non-functioning of software can cause direct damage - no software integrity is maintained automatically. Applications programs with mammoth-illness are too big to be transferred over the network just to try to run them once and then abandon them, because they have proved to be different from what end-user thought they would be, or were behaving badly in other ways, or even did not work at all whereby program is said to be incompatible with hardware. It could be said that software does not maintain conformity against changing environments. The EXE-mode is a remainder from the stone-age of computer technology - programs are distributed by copying them as such over the network and exactly the same copy of a program may be kept in many different stores making the use of disk storage resource totally inefficient. Software is not efficient and economic in using resources.

Maintenance

Software has usually provided a version for many different execution environments, operating systems or device platforms. There can be different versions offered even for exactly the same execution environment seemingly just for fun, or with small improvements to their versions. It is the responsibility of the maintenance person of an environment to keep different versions counted and available. Updating software is an established practice in many enterprises today - software itself does not maintain software consistency against version changing.

Software industry

Investments in software found unpleasant afterwards are common. Consequently, there is immorality in not following end-user licence agreements, and global pirate manufacturers culture exists. This is directly decreasing the productivity of software production and content providing - software does not maintain licensing and authorising for them.

Society

It is common that people trade products and services. We like the situation when products and services are advertised or offered to us, so we can make our choice to get them after we have seen whether the product or service is such as we want it to be. We can also lease the product, and with services, we like to pay only for their use, for example, as we pay when we use dentistry services. The situation with software products and information services should be the same as depicted with usual consumables. Software products and information services may be provided by content providers and service providers, as is the case today. These software products and information services should be treated just like usual consumables - according to the laws and rules set by society, and equally, according to contracts between end-user and content or service providers. The end-user should see himself surrounded by useful and profitable software applications, which can be deployed just by understanding the business function of a product - as we understand the function when we go e.g. to the dentist's. All those applications should be provided with good maintenance and guarantees the way we demand these on usual products.

The new approach

The technology to be used should have a self-configurability feature - products should maintain themselves when used in different circumstances. The end-user should not have to worry about whether the software used has to be updated, or even whether the software and hardware resources in his terminal equipment are updated and working - the technology must provide the maintenance without any end-user intervention. The end-user should always have, even unknowingly, updated software and even updated hardware support - this may also be a question of contracts. Furthermore, does the end-user have to be bound to some individual device to maximise the deployment of technology, as is the case with our desktop computer today? This does not have to be the case; instead, the end-user should be able to deploy the application with his personal preferred configuration wherever he happens to be located and whatever device he happens to use.

Let us take an example: the end-user would like to use his banking services for checking the account balance - therefore he just takes some terminal equipment, a public kiosk terminal, a cell phone or whatever compatible device happens to be near, and will be able to reach and deploy the service wanted with personal settings for the device type of the terminal and with personal settings for the service used - this, moreover, with great security and usability.

Summary

In Table 2, the user requirements above are gathered together and then developed for the table. Requirements are grouped according to the role of the user behind the request. To follow requirements further, they are labelled as PEx, where the x is the number of the item.

Table 2. List of preferred user requirements with reference label.

Role	Need	Lb.
End-user	End-user can easily reach and deploy a software product or information service by using any compatible terminal equipment, according to a contract.	PE1
	End-user gets his personal profiles for the equipment.	PE2
	End-user gets his personal profiles for a service.	PE3
(continues)	End-user has data and access security maintained, according to a contract.	PE4
The table continues.		

Continuing Table 2.

Role	Need	Lb.
End-user *(continuing)*	End-user will be provided a vision of the information environment that offers easy-to-use, and even profitable applications and services.	PE5
Society	Software products or information services are considered as usual consumables (e.g. in relation to marketing regulations, product liability or data security).	PE6
Device provider	Device provider can provide compatible terminal equipment with maintenance according to a contract.	PE7
Service provider	Service provider can provide rich applications types the way end-users are provided with today, but in a maintained mode.	PE8
	Service provider can provide software products or information services with maintenance according to a contract.	PE9
	Service provider can do business by trading software products and information services with contracts.	PE10
Content provider	Content provider can maintain the authorising and licence information with provided software products or information services according to a contract.	PE11

2.2 User's requirements as threshold

To clarify the specifications and requirements, which may concern the distributed system, the theory, standards, and techniques used with practical systems of the distributed computing area will be examined.

2.2.1 Theoretical user requirements

In this subsection the topic of the requirements for distributed system presented in the theory will be addressed. Grouping of those requirements is presented as well. Colouris et al. have analysed user requirements for practical distributed systems [16]. They have placed requirements in three main groups (these do not reflect any of Kano's groups). The requirements are combined in Table 3. With the table, Kano's threshold requirements are beginning to be reached.

Table 3. Theoretical user requirements by Colouris et al.

Category	Requirement	Notes
Functionality - what the system should do for users.	Economy and convenience	The economy and convenience in sharing hardware resources and information.
		Support for a new kind of functionality of applications, such as CSCW (Computer Supported Co-operative Work).
		Easy API for application programmers.
	Transition	Adapt existing investments for software and systems.
	Migration	Adapt existing applications.
		Adapt existing operating systems.
The table continues.		

Continuing Table 3.

Category	Requirement	Notes
Reconfigurability - the need for a system to accommodate changes without causing disruption to existing service provision.	Reconfigurability	The ability to accommodate short-term changes e.g. failure, load or replacement of a component.
		The ability to accommodate medium-to-long terms evolution of software, hardware or infrastructure.
Quality of service - user perception of a system.	Performance	A fast and consistent response to the user's interaction and the impact of communication, when it is presented on the user's screen.
	Reliability and availability	A measure of how small is the likelihood of the system deviating from behaving as it was designed to do.
	Security	Protection of users, data and systems against unauthorised, maybe malicious access.

2.2.2 Standards

There are two kinds of standards organisations: 1) international treaty-based standards organisations whose task is to develop standards to enable international trade - this include, most notably, ISO (International Standardisation Organisation) and ITU-T (International Telecommunication Union - Telecommunication standardisation sector), and 2) industrial consortia to be formed to develop and promote particular standards. Also, there are standards which have emerged from the market place through having achieved a certain level of market penetration; examples include the "IBM-compatible" personal computer and the Java programming environment.

Standards produced by recognised standards organisations are referred to as de jure because their status is defined via treaties. In contrast, standards that emerge from the market place are referred to as de facto standards. Standards produced by consortia such as OMG (Object Management Group) are also classed as de facto, because these consortia have no international status as standards organisations.

A number of standardisation activities are addressing the issues of open distributed processing. These are [12]:

- OSI (Open Systems Interconnection) reference model adopted by ISO is a conceptual model for seven-layer protocol stack for inter-net working. The OSI Reference Model was adopted in order to encourage the development of protocol standards that would meet the requirements of open systems.

- RM-ODP [17] is a joint standardisation activity by both ISO and ITU-T. The standard provides an object-oriented system specification methodology based on the concept of viewpoints together with a vocabulary and grammar for describing a distributed system from each viewpoint.

- CORBA (Common Object Request Broker Architecture) is a standardisation activity promoted by OMG [18]. OMG is a non-profit organisation sponsored by several of IT (Information Technology) and telecommunications companies with the specific aim of promoting an open object-oriented framework for distributed computing. The particular goal of CORBA is to provide the mechanisms by which objects transparently interact in a distributed environment.

- The Open Group' DCEs (Distributed Computing Environment) is a standardisation activity sponsored by the Open Group. The Open Group, like OMG, is a non-profit organisation sponsored by a number of IT and telecommunications companies with the aim of promoting computing standards. DCE is an operating system-independent and network-independent software platform to achieve open distributed processing. In contrast to RM-ODP and CORBA, DCE supports simple client-server architecture - it does not support object-oriented abstractions. It is not considered in this document.

When RM-ODP was developed in the ANSA-project (The Advanced Network System Architecture Project) the results were concurrently implemented in the FlexiNet project and in its extension for the mobile object computing - the FollowMe project [19]. Another example of the specialisation of the architectures above is the TINA (Telecommunication Information Networking Architecture) which is currently being developed by the TINA consortium (TINA-C) [20, 21] consisting of a number of leading telecommunications companies. This architecture is based closely on RM-ODP, but with a specialisation to meet the particular demands of the telecommunications industry.

ISO/OSI Reference Model

The OSI Reference Model was adopted in order to encourage the development of protocol standards that would meet the requirements of open systems. Layers are explained in Table 4. The table is combined from the sources [2] and [22]. The last column shows the TCP/IP-protocol family interpretation of layers [23]. Borders between layers when compared with the pure OSI Reference Model and TCP/IP-stack must be considered as a conceptual presentation.

Table 4. OSI protocol summary.

Layer	Description	Examples	
		OSI	TCP/IP
Application (7)	Protocols that are designed to meet the communication requirements of specific applications, often defining the interface to a service.	X.400, X.500	HTTP, FTP, Telnet
The table continues.			

Continuing Table 4.

Layer	Description	Examples	
		OSI	TCP/IP
Presentation (6)	Protocols at this level transmit data in a network representation that is independent of the representations used in individual computers, which may differ. Encryption is also performed in this layer, if required.	ASN.1, encryption	ASCII
Session (5)	At this level communication between processes is established and error recovery is performed. It is not required for connectionless communication.		RPC
Transport (4)	This is the lowest level at which messages (rather than packets) are handled. Messages are addressed to communication ports. The protocol in this layer may be connection-oriented or connectionless.		TCP, UDP
Network (3)	Transfers data packets between computers in a specific network.	X.25	IP
Data link (2)	Responsible for error-free transmission of packets between computers which are directly connected.	HLDC, CSMA/CD	
Physical (1)	Circuits and hardware that drive the network.	X.21	

Shueu et al. [24] point out that it seems like everything that is not in the lower six layers has been pushed into the application layer and suggests a modification of OSI reference model for distributed information systems. The suggested model is presented in Table 5.

Table 5. Possible modification of OSI reference model by Shueu et al.

OSI perspective	Layer	Information systems perspective
User oriented	Ultimate user (end-user)	
	Information system levels	Information and data services
	Application	Syntactic user interfacing
	Presentation	
	Session	
End-to-end connection oriented	Transport	Network transport services
	Network	
Point-to-point link oriented	Data link	
	Physical	

ISO/RM-ODP Reference Model

The aim of RM-ODP is to "enable the development of standards that allow the benefits of distribution of information processing services to be realised in an environment of heterogeneous IT resources and multiple organisational domains". RM-ODP itself does not prescribe particular standards for open distributed processing. Rather, it provides a framework (in terms of architectural concepts and terminology) to enable specific standards to emerge. RM-ODP should thus be considered a meta-standard for open distributed processing. It is therefore essential in the definition of RM-ODP to be sufficiently generic to enable a range of standards to be accommodated within this framework. The main value of RM-ODP is thereby in providing a common set of concepts for the broader field of open distributed processing. The interested reader can turn to the specific literature for a more detailed introduction [12]. It is worth noting that many of the ideas described below were developed initially in the ANSA project [25]. This work also produced a prototype platform, ANSAware, as a partial implementation of the ANSA architecture. Major concepts in RM-ODP are:

- OO approach,
- viewpoints and viewpoint models,
- distribution transparency, and
- RM-ODP functions.

Object-oriented approach is adopted by RM-ODP for the specification of distributed systems. The arguments in favour of this approach are summarised below [12]:

- Encapsulation is the natural approach to developing distributed applications.
- Data abstraction provides a complete separation between specification and implementation of objects, thus enabling design in terms of interfaces.
- The separation between specification and implementation also enables the system to evolve naturally by allowing an object to be replaced by an alternative implementation (as long as the interface remains the same).
- Object-oriented systems are completely extensible in that new classes and objects can be added at any time.
- The concepts of implementation and interface inheritance support the reuse of code and interfaces respectively.
- The concept of sub-typing provides flexibility in selecting services in a distributed environment.

Viewpoints are used to partition a distributed system specification. RM-ODP defines five viewpoints: the Enterprise, Information, Computational, Engineering and Technology Viewpoints. Each viewpoint has a corresponding viewpoint model and is a complete and self-contained description of the required distributed system targeted towards a particular audience with the proper terminology. Viewpoints are projections on to the underlying system. In Figure 3 [12] there is an illustration of the example of correspondence between the computational viewpoint and the engineering viewpoint.

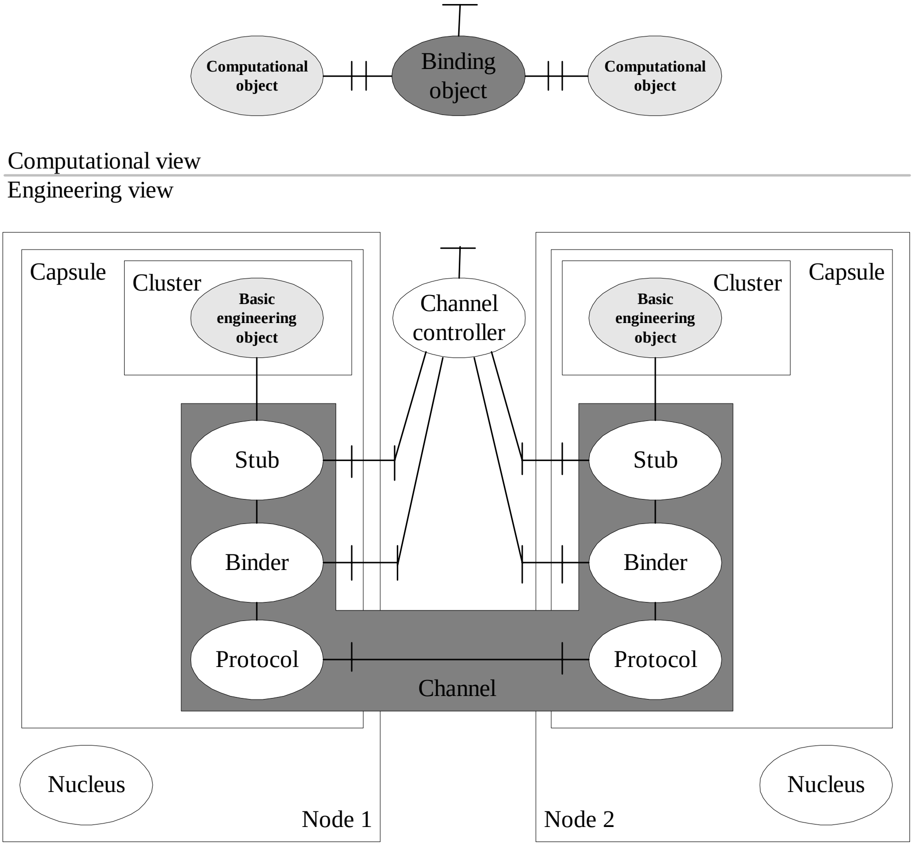


Figure 3. Example of correspondence between computational and engineering viewpoints.

The viewpoints and models are given in Table 6 [12].

Table 6. RM-ODP viewpoints and models.

Viewpoint	What does it address	Modelling concepts
Enterprise	Business concerns	Contracts, agents, artefacts, roles and so on.
Information	Information, information flows and associated processes.	Objects, composite objects, schemata (static, dynamic, invariant)
Computational	Logical partitioning of distributed applications	Object, interface, environmental contracts and so on.
Engineering	Distributed infrastructure to support applications.	Stubs, binders, protocol objects, nodes, capsules, clusters and so on.
Technology	Technology procurement and installation.	Implementation, conformance point, IXIT

Distribution transparency is the ability to mask out problems occurring in the distributed environment. A high level of distribution transparency therefore implies that the programmer does not need to be aware of the distributed nature of the system. This can be highly beneficial because it reduce the complexity of distributed programming. The approach in RM-ODP is to support selective transparency whereby the programmer can elect a given level of transparency. As mentioned above, the requirements for distribution transparency are expressed in the Computational Model as part of the environmental contract. The Engineering Model is then responsible for meeting the desired level of transparency by employing the required number of transparency functions.

A summary of the distribution transparencies defined in RM-ODP is given in Table 7 [12]. It should be noted that this list is not intended to be exhaustive; further transparencies can therefore be introduced for particular application domains.

Table 7. Distribution transparencies (Transp.) in RM-ODP.

Transp.	Concern	Effect
Access	Means of access to objects	To mask out differences in data representation or invocation.
Location	The physical location of objects	To enable objects to be accessed by a logical name.
Failure	The failure of an object	To mask out this failure from the user through an appropriate recovery scheme.
Migration	The movement of objects	To mask out the fact that an object has moved.
Relocation	The movement of objects involved in existing interactions	To mask out the fact that an object has moved from the other interacting objects.
Replication	Maintaining replicas of objects	To hide the mechanism required maintaining consistency of replicas.
Persistence	To maintain persistency of data across interactions	To mask out the mechanisms required maintaining persistency of data.
Transaction	Maintaining consistency of configurations of objects	To hide the mechanisms required maintaining consistency of configurations.

RM-ODP functions fulfil a number of different purposes in RM-ODP. For example, the standard defines a trading function to act as a broker for services in the distributed environment. The trading function enables objects to make interfaces available by exporting a service offer to the trader. An object wishing to interact with a service interface must import the interface by specifying a set of requirements in terms of service type and related constraints. This will be matched against the available services and a suitable candidate selected. It needs to be also noted that any number of traders can exist in a given distributed system and these may be linked to allow access to services in different domains. The complete set of functions defined in RM-ODP is summarised in Table 8 [12].

Table 8. Functions in RM-ODP.

Area	Specific function	Concern
Management	Node management	Manages resources of a node, including threads and clocks.
	Object management	Check-pointing and deletion of objects
	Cluster management	Checkpoints, recovers, migrates, deactivates or deletes clusters.
	Capsule management	Creates instances, checkpoints or deactivates associated clusters and deletes capsules.
Coordination *(continues)*	Event notification	Records and makes available event histories.
	Check-pointing and recovery	Overall coordination for check pointing and recovery of clusters.
The table continues.		

Continuing Table 8.

Area	Specific function	Concern
Coordination *(continuing)*	Deactivation and reactivation	Overall coordination for deactivation and reactivation of clusters.
	Groups	Coordinates the interaction of objects in multiparty binding.
	Replication	Coordinates interaction in a replica group.
	Migration	Coordinates the migration of clusters between capsules.
	Interface reference tracking	Maintains information on interface references.
	Transaction	Coordinates set of operations to ensure visibility, recoverability and permanence.
	Storage	Provides the storage for data.
Repository	Information organisation	Supports querying and modification of information schema and data.
	Relocation	Supports relocation transparency through a relocator object.
	Type repository	Maintains a repository of type specifications and relationships.
(continues)	Trading	Supports advertising and discovery of interfaces.

The table continues.

Continuing Table 8.

Area	Specific function	Concern
Repostitory *(continuing)*	Access control	Prevents unauthorised interaction on interfaces.
Security	Security audit	Monitors and collects security information.
	Authentication	Confirms the identity of an object
	Integrity	Prevents unauthorised creation, alteration or deletion of data.
	Confidentiality	Prevents unauthorised disclosure of information.
	Non-repudiation	Prevents the denial of an object from having participated in an interaction.
	Key management	Manages cryptographic keys.

OMG/CORBA

CORBA [26, 27, 28] is OMG's open, vendor-independent architecture and infrastructure that computer applications use to work together over networks. Using the standard protocol IIOP (Internet Inter-ORB Protocol), a CORBA-based program from any vendor, on almost any computer, operating system, programming language, and network, can interoperate with a CORBA-based program from the same or another vendor, on almost any other computer, operating system, programming language, and network.

2.2.3 State-of-the-practice

In this subsection, the existing distribution concepts and techniques will be discussed. The techniques covered here are restricted so that the selected techniques will, as far as possible, represent all the techniques of the distributed computing in all the categories in the market today. Only the techniques that are most relevant for the purpose of this study will be concentrated on.

There is no perfect system on the market, and such may never come to exist. Different concepts of the distributed services in the network 1) realise different sets of the demanded requirements, 2) consider different sets of technical aspects that different environments may set up, 3) still respect the subjective opinion of the inventors or the developers of these techniques. Especially, an attempt will be made to find out the user requirements behind the design of every particular system and, on the other hand, the practical technical features by which the user requirements have been demanded.

The UNIX X-Window system deploys very traditional fat-client/server techniques. The Terminal Server solutions, which consider Citrix MultiFrame and Microsoft Terminal Server, introduce the server-based computing architecture model. Distributed application execution environments (also called Application server) are represented by the Oracle Application Server [29]. It introduces an Internet-based computing model. However, we do not include this technique in our presentation here; instead, we do include the very new technique of Microsoft.NET. The common feature between these techniques is that the application can be extended to be used through network by an Internet-browser such as Netscape Navigator or Internet Explorer.

The .NET-concept has adopted many features introduced by Jini before, but may still add some extra value on the original concept of Jini. This is because of its ability to enable the applications of the large and rich Microsoft software family to inter-operate through distributed environments by distributing some part of them. It is assumed here that the common features of Jini and .NET are the same that will be needed in our solution as well. As an example of the common features, the concept of proxy needs to be mentioned. Also, many applications which are intended to be used to make group-work more efficient, i.e. CSCW-software such as Lotus Notes or Microsoft Exchange, include many features by which the user can distribute their own application, therefore these can be considered to be application servers of some kind. The .NET-technique also extends the Microsoft-application family with an ability to distribute some part of them through network, and these distributed parts will exist embedded in an Internet-browser window, as Java-applets do. The ASP (Active Server Page) technique is used for this purpose. Furthermore, we would like to introduce two more techniques: Mobile Station MExE (Application Execution Environment), and OSGi (Open Services Gateway Initiative). These techniques are directed to different sectors of the market, MExE for mobile terminal and OSGi to distribute home services. OSGi resembles the solution introduced in the LONTONEXTG student project [10].

Distributed software components and objects are the building parts of distributed applications. Therefore, they must be made manageable. Someone may think that the existing distributed object techniques, like CORBA, already provide the answer to the specific research problem under scrutiny here. However, although these objects are significant, they do not solve the management problem in question.

UNIX's X-Window System

The standard protocol for graphically based UNIX-based applications is the X11 protocol [30]. The X-Window System environment is based on the typical distributed processing model - client/server. A fundamental concept of the X-Window System is the separation of processing the algorithm (X Client) from the handling of user interactions on the display, keyboard, and mouse (X Server). These are two different programs, which may run on the same physical processor or on two different systems. If they are separated on distinct machines, they require a communication link.

The X Server application, which typically resides on the client side, provides the following functions:

- manages the output on the monitor,

- manages the keyboard and mouse input,

- manages the input/output queues,
- handles the communication between the X Server and the X Client,
- stores off-screen data, and
- downloads fonts.

The presentation of an X-Windows-based application is dependent on the X Server. The X-Windows environment is more than just a network protocol; it’s also an application development paradigm. Applications are developed using the X-Windows toolkit, just as Windows-based applications are developed using the Windows SDK (Software Development Kit). Because the X-Windows protocol represents fat client architecture, the client requires a considerable amount of processing power.

Server-based computing by Citrix

Citrix Systems, Inc. [31] has been developing thin-client architectures and application servers. Two core technologies form the foundation for Citrix's application server solutions: 1) ICA (Independent Computing Architecture), which centralises all application processing on the server, delivering cost and performance benefits through single-point application management, reduced bandwidth requirements, support for virtually any platform or device, and improved security, and 2) MultiWin, which allows multiple concurrent users to log on and run applications in separate, protected Windows sessions from a single server. Server-based computing is a model, in which 100% of deployment, management, support and execution of applications are carried out on a server. The model uses a multi-user operating system and a method for distributing the presentation of an application's interface to a client device. With server-based computing, client devices, whether "fat" or "thin", have instant access to applications via the server. Applications are centrally managed and can be accessed by users, without having to rewrite them.

Table 9 presents a list of some challenges that can be seen accomplished by server-based computing [31]:

Table 9. Challenges of enterprise-wide application deployment.

Challenge	Note
Management	Managing and supporting users in a timely and cost-effective manner.
Access	Extending access to business critical applications to dispersed users - regardless of connection, location or device.
Performance	Ensuring exceptional application performance.
Security	Providing tight security for enterprise level computing.

Microsoft Terminal Server

With Windows NT Server 4.0 operating system, Microsoft published a new product - Windows-based Terminal Server (code-named Hydra) adds UNIX-like multi-user capabilities and support for thin-client Windows-based Terminals to the Windows NT Server 4.0 operating system [30]. MultiWin, the technology licensed by Citrix to Microsoft to create the Terminal Server, enables multiple users to simultaneously access applications running on a server.

Terminal Server environment is, by definition, a thin-client architecture where all application processing occurs centrally on the server. Terminal Server consists of three primary components, viz. of:

- the Windows NT Server multi-user core,
- the RDP (Remote Desktop Protocol), and
- the "super-thin" Windows-based client software.

Based on a multi-user server core that provides the ability to host multiple, simultaneous client sessions on Windows NT Server, Terminal Server can provide remote access to a 32-bit Windows NT–based desktop for a number of

Windows-based and non-Windows-based hardware. Standard Windows-based applications do not need modification to run on the Terminal Server and almost all Windows NT–based management infrastructure and technologies can be used to manage the client desktops.

RDP is an essential component of the Terminal Server. It allows a super-thin client to communicate with the Terminal Server over the network. This protocol is based on the ITU T.120 protocol, an international-standard multichannel conferencing protocol already an integral component of the Microsoft NetMeeting conferencing software product. Encrypted sessions are also supported in this protocol. Integrating the third-party add-on Citrix MetaFrame, Terminal Server can be expanded to communicate to a variety of thin-client devices. The ICA protocol stack can also be run on a traditional "fat client" or even through ICA-enabled Web browsers. With this flexibility, organisations can achieve a critical goal - to provide application access to any clients regardless of their platform.

Microsotf.NET

Microsoft.NET remoting provides a framework that allows objects to interact with one another across application domains. The framework provides a number of services, including activation and lifetime support, as well as communication channels responsible for transporting messages to and from remote applications. Some details of these services have been combined in Table 10.

Formatters are used for encoding and decoding the messages before they are transported by the channel. Applications can use binary encoding where performance is critical, or XML (Extended Mark-up Language) encoding where interoperability with other remoting frameworks is essential. All XML encoding uses the SOAP (Simple Object Access Protocol) in transporting messages from one application domain to the other. With SOAP, data stream is transported to the target URI (Universal Resource Identifier) using the HTTP (HyperText Transfer Protocol). The interested readers are encouraged to continue with the source reference [11].

Table 10. Services of the .NET-framework.

Service	Explanation
Remote Objects	One of the main objectives of any remoting framework is to provide the necessary infrastructure that hides the complexities of calling methods on remote objects and returning results. Any object outside the application domain of the caller should be considered remote, even if the objects are executing on the same machine.
Proxy Objects	Proxy objects are created when a client activates a remote object. The proxy object acts as a *representative* of the remote objects and ensures that all calls made on the proxy are forwarded to the correct remote object instance.
Channels	Channels are used to transport messages to and from remote objects. *HTTP Channel* transports messages to and from remote objects using the SOAP protocol. All messages are passed through the SOAP formatter, where the message is changed into XML and serialised, and the required SOAP headers are added to the stream. The binary formatter can also be specified, which results in a binary data stream. *TCP Channel* uses a binary formatter to serialise all messages to a binary stream and transporting the stream to the target URI using the TCP protocol.
Activation	The remoting framework supports server and client activation of remote objects. Server activation is normally used when remote objects are not required to maintain any state between method calls. It is also used in cases where multiple clients call methods on the same object instance and the object maintains state between function calls. On the other hand, client-activated objects are instantiated from the client, and the client manages the lifetime of the remote object by using a lease-based system provided for that purpose.
The table continues.	

Continuing Table 10.	
Service	Explanation
Object Lifetime with Leasing	Each application domain contains a lease manager that is responsible for administrating leases in its domain. All leases are examined periodically for expired lease times. Using leases to manage the lifetime of remote objects is an alternative approach to reference counting, which tends to be complex and inefficient over unreliable network connections. Although one could argue that the lifetime of a remote object is extended longer than required, the reduction in network traffic devoted to reference counting and pinging clients makes leasing a very attractive solution.

MExE

MExE (Mobile Station Application Execution Environment) is a wireless protocol that is designed to be incorporated into smart mobile phones [32, 33]. MExE:

- aim is to provide a comprehensive and standardised environment on mobile phones for executing operator or service provider specific applications,
- is designed as a full application execution environment on the mobile terminal,
- builds a Java Virtual Machine into the client mobile phone,
- is designed to be used to provide sophisticated intelligent customer menus and also facilitate Intelligent Network services,
- plans to integrate mobile phone location services,
- supports a wide range of man-machine interfaces such as voice recognition, icons and softkeys, and

- supports device profiles by defining the "classmark" properties of terminal devices.

MExE shares several similarities with the WAP (Wireless Application Protocol) [34] in that both protocols have been designed to work with a range of GSM (Global System for Mobiles) mobile network services from SMS (Short Message Service) to GPRS (General Packet Radio Service) and later with UMTS (Universal Mobile Telecommunications System). Whereas WAP incorporates some scripting, graphics, animation and text, MExE allows full application programming. This necessitates the need for MExE to include a strict security framework to prevent unauthorised remote access to the user's data. Because programming and running Java applications requires significant processing resources on the mobile client, it is primarily aimed at the next generation of powerful smart telephones. On the other hand, MExE terminals can also include today’s regular telephones, because MExE incorporates a capability indication method called classmarks. MExE classmarks define the MExE-related services that a particular terminal supports. There will be classmarks that match and those that exceed WAP functionality. The MExE mobile client can inform the MExE server of its classmark and therefore its capabilities.

Development of the initial MExE protocols is being carried out in SMG4. SMG4 is the GSM standards setting body of the European Telecoms Standards Institute (ETSI) [35], a standardisation group responsible for GSM data. Supporters of this work include Motorola, Nokia, Lucent Technologies and Nortel (with the concept Nortel Orbitor smart telephone in line with the MExE concept).

MExE will be available later than WAP because the processing power to run the Java applications is not currently available in mobile terminals.

OSGi

The OSGi [36] is a Non-Profit corporation organised in the USA, and supported by 60 companies worldwide. It defines a set of APIs and provides a sample implementation of service gateway architecture. This services gateway is inserted between the external network and internal network and devices. The Services Gateway inserts a new value point into home networks that will facilitate the development and deployment of a wide range of advanced network based services. Service providers will be delivering just-in-time value added services to this services gateway and this gateway will provide a service distribution, integration and management point in a SOHO / ROBO (Small Office/Home Office and Remote Office/Branch Office) or residence. OSGi is targeted not only towards the residential gateway marketplace, but towards also the SOHO/ROBO market.

Mixture of Distributable Objects

Re-usable software components and objects [37] are the key factor to increase In-house resources of software houses (or enterprises) and as a consequence, add productivity of the software production. There will be more and more application software built by using existing software components with re-usability in mind. These components may need modifications, and modifications further increase resources.

There has been a development from simple re-usable software components to building up component libraries for different purposes. Yet, these libraries are developed so that an individual component in the library can be easily extracted and then modified or customised to match the particular needs. Examples include Java Beans and OCX-controls (Object Linking and Embedding - OLE custom controls). On the top of this development hierarchy there stand the high power development tools like JBuilder or Visual Basic. For the reader, the source [38] includes the full collection of information needed to examine this issue further.

After the copy and paste method to re-use components had been passed, the next generation techniques were appearing. The software componentisation was developed so that it was possible to combine applications from living software objects, i.e. from objects that were initialised beforehand, and were already working somewhere in the network. Their services only need to be used. The services are known by some interface-techniques and we do not need to know how the object actually produces the service. The term remote interface is used in this context. It is possible to speak about server components - a distributed object acts like a server. Examples include JavaRMI, CORBA and DCOM-models. The source [39] includes an all-out comparison of these techniques.

Improvements of availability and usability to deploy these server components have been made by collecting them together in containers. The container structure assists in serving the objects. Examples include MTS (Microsoft Transaction Server) and EJB (Enterprise JavaBeans). CORBA has been supported in version 3 by the CCM (Corba Component Mode) [40], and it is compatible with the EJB-model in such a way that CCM can deploy the Java Beans. Arrangements of container structure can easily support different transparency features such as transaction, relocation, failure and so on. However, the access and location transparencies demand (so called network transparencies) will not be fulfilled. It is possible to serve resources as they are in a resource pool, making the resource pooling possible. The source [41] includes a very detailed comparison of these container structures.

Still, there is the problem of how the different object models can inter-operate. Some gateway arrangements do exist, like DCE/CORBA [42], which connects the object-oriented world to the traditional client-server world.

In the distributed software environment, application programs become more and more based on the components (or living objects). All problems that exist with deployment of distributed objects will exist on the application level as well as on the management level of software.

A new paradigm in Distributed Computing - Jini

Jini's ability to support spontaneously created, self-healing communities of services is based around five key concepts. The concepts are presented in Table 11 [1].

To use a service, a person or a program locates it using the lookup service. The service's object is copied from the lookup service to the requesting device where it will be used. The lookup service acts as an intermediary to connect a client looking for a service with that service. Once the connection is made, the lookup service is not involved in any of the resulting interactions between that client and that service.

Table 11. The five key concepts of Jini.

Concept	Explain
Discovery	Discovery is the process used to find communities on the network and join them. Discovery is responsible for the spontaneous community-building properties of the system.
Lookup	Lookup fulfils the role of a directory service within each Jini community, and provides facilities for searching and finding services that are known within a community.
Leasing	Leasing provides Jini's self-healing nature.
Remote Events	Remote events are the paradigm Jini uses to allow services to notify each other of changes in their state. Because lookup is a service, it can use remote events to notify interested parties when the set of services available to a community has changed.
Transactions	Transactions are mechanism for allowing computations that may involve multiple services to reach a "safe" state. Transaction model helps guard against the failures in a distributed system, helps address the concurrency problems posed by distributed systems and provide services with a resilience to network failures.

The Java programming language is the key to making Jini technology work. Devices in a network employing Jini technologies are tied together using Java RMI (Remote Method Invocation) [43]. The discovery and join protocols, as well as the lookup service, depend on the ability to move Java objects, including their code, between Java virtual machines. Jini's leasing and transaction mechanism provide resilience in a dynamic networked environment.

The Jini design centre

The Jini design centre is the set of areas that the Jini designers felt were the most important to focus on (Table 12) [1]. The table will be used when the presented solution will be evaluated in Chapter 7.

Table 12. The Jini design centre.

Point	Explanation
Simplicity	Jini adds only a thin veneer on Java to allow devices and services on the network to work with each other more easily. Jini is about how services connect to one another - not about what those services are or what they do or how they work. Jini services can be written in a language other than Java; the only requirement is that there exists, somewhere on the network, a bit of code that is written in Java that can participate in the mechanisms Jini uses to find other Jini devices and services. Everything - even a device such as a scanner or printer or telephone - is really a service. Hardware devices can be understood in terms of the interfaces they present to the world.
Reliability	Jini provides the infrastructure that allows the services to find and use one another on a network. The concept of a name server like the Internet's DNS (Domain Name Service) or the LDAP (Lightweight Directory Access Protocol) or ISO's X.500 directory service protocol are usually considered to carry this responsibility. The spontaneous networking abilities of Jini mean that the configuration of the network can be changed without involving system administrators, and the ability to take advantage of devices previously unknown in a Jini community means that there is no need for driver or software installation to use a particular service (other than the installation of the core Jini software itself, of course).

The table continues.

Continuing Table 12

Point	Explanation
Scalability	Groups of Jini services join in co-operating sets. In Jini, these groups of services are called communities; all services in a community are aware of each other and able to use each other. Jini services band together to form communities. Jini addresses scalability through federation. The size for a single Jini community is about the size of a workgroup - the number of printers, PDAs, cell telephones scanners, and other devices and network services, needed by a group of people.
Device agnosticism	Jini is agnostic with regard to devices. It means that Jini is designed to support a variety of entities that can participate in a Jini community. These "entities" may be devices, or software or some combination of both; in fact, it is generally impossible for the user of one of these things to know which it is. To use something whether that something is hardware or software, only the interface it presents must be understood. Jini does not require that the device or service be written in or understands Java, all that is required is that some Java-speaking device be willing to act as a proxy on behalf of the Java-challenged device or service.

2.2.4 Harvest of the user requirements as Threshold

The user requirements to be set as Kano's threshold values are listed in Table 13. The sources of every item are listed, too. To follow requirements further, they are labelled as Trx, where the x is the number of the item.

Table 13. Threshold u*ser requirements.*

Category	Requirement	Notes	Source	Lb.
Functionality - what the system should do for users.	*Access*	Extending access to applic. to dispersed users - regardless of connection, location or device.	Table 10	Tr1
	Economy and convenience	The economy and convenience in sharing hardware resources and information.	Table 3	Tr2
		Support for a new kind of functionality of applications (e.g. CSCW).	Table 3	Tr3
		Easy API for application programmers.	Table 3	Tr4
	Transition	Adapt existing investments.	Table 3	Tr5
	Migration	Adapt existing applic.	Table 3	Tr6
		Adapt existing operating systems.	Table 3	Tr7
The table continues.				

Continuing Table 13.				
Category	Requirement	Notes	Source	Lb.
Reconfigurability - the need for a system to accommodate changes without causing disruption to existing service provision.	*Reconfigurability*	The ability to accommodate short-term changes e.g. failure, load or replacement of a component.	Table 3	Tr8
		The ability to accommodate medium-to-long-term evolution of sw, hw or infrastructure.	Table 3	Tr9
Quality of service - user perception of a system.	*Management*	Managing and supporting users in a timely and cost-effective manner.	Table 10	Tr10
	Performance	Ensuring exceptional application performances.	Table 10	Tr11
		A fast and consistent response to the user's interactions.	Table 3	Tr12
	Reliability and availability	A measure of how small is the likelihood of the system deviating from behaving as it was designed to do.	Table 3	Tr13
(continues)	*Security* *(continues)*	Providing tight security for enterprise level computing.	Table 10	Tr14
The table continues.				

Continuing Table 13.

Category	Requirement	Notes	Source	Lb.
Quality of service *(continuing)*	*Security (continuing)*	Providing tight security for enterprise level computing.	Table 10	Tr14
		Protection of users, data and systems against unauthorised, maybe malicious access.	Table 3	Tr15

2.3 Mini-concepts

Mini-concepts found are presented in Table 14. The source of every Mini-concept is also presented. Mini-concepts are based on the study of standards in Subsection 2.2.2 and State-of-the-practice in Subsection 2.2.3. Not all the Mini-concepts have to be presented, some are pointed in order to establish the Big-picture. The more there are Mini-concepts presented and handled later in QFD-analysis, the more certain it is that the solution presented is correct one. However, the number of the items in QFD-matrix is limited, for practical reasons. The following presents the reference labels again.

Table 14. Some Mini-concepts (M-c) in realised distributed systems.

Mini-concept	Source technique	Lb.
Program logic is processed only on client-side	Fat-client/server computing	M-c 1
Program logic is processed only on server-side	Server-based computing	M-c 2

The table continues.

Continuing Table 14.

Mini-concept	Source technique	Lb.
Services join communities represented by the directory service	Jini Join-protocol	M-c 3
Directory services contain more directory services banding up the federations	Jini Lookup service	M-c 4
Directory service has a capability of containing simple attributes	X.500 directory service	M-c 5
Directory service has a capability of containing a client program logic	Jini Lookup service	M-c 6
Directory service has a capability of containing a description of client program logic	UPnP (not included)	M-c 7
Server-side program logic and client-side program logic can use different executable-code modes	Jini, .NET	M-c 8
Thin or super-thin client with event and user screen transfer	Server-based computing	M-c 9
Movable description of client-side program logic	XML	M-c 10
Movable object contains client-side program logic with run-time attributes	Jini's proxy, the proxy of the .NET	M-c 11
Program logic can be defined to run on client-side or server-side at the programming time	ASP, Jini	M-c 12

The table continues.

Continuing Table 14.

Mini-concept	Source technique	Lb.
Services form into communities	Jini Discover-protocol	M-c 13
Client-side program logic run-time condition is to be initialised when application is starting	Jini's proxy, Server-based computing	M-c 14
Client-side program logic is secure against modification	JVM	M-c 15
Strategies to maintain resources can be set by means of management policy	RM-ODP	M-c 16
Application program logic can be programmed without programming the support for distribution of the code	RM-ODP Distribution transparency, .NET	M-c 17
The distribution transparency is arranged by a binding object	RM-ODP's binding objects	M-c 18
Different capabilities of terminal devices are classified by the device profiles	MExE	M-c 19
The flexible channel structure enables the communication	RM-ODP, .NET	M-c 20
The proxy object represents the application in the distributed environment	Jini, .NET	M-c 21

2.4 Summary

In the past chapter the requirements specification of the system was completed. The Threshold attributes of the Kano's Model in the Requirements Specification were thereby discussed. The Threshold requirements found are listed in Table 13, and they will be used accompanied by the Excitement and Performance requirements of the users presented in Table 2 to drive the selection of the system properties as the study progresses. By this, the system quality is ensured before actually doing anything. Thereby, objective quality has been discussed.

Apart from the user requirements, there is the other group of requirements to take into account - the requirements that are set by the existing technical environment. Part of these technical requirements is due to the standardisation, and part is due to the requirements of compatibility with the mainstream. The standardisation factors of de jure and de facto have been introduced. Furthermore, there are requirements wihch are a consequence of some technical limitations that must be considered. The technical requirements will be sharpened and defined further as the development of the system progresses, but the requirements of the users will still drive the selection of the technical features.

It was found out that the RM-ODP reference model is actually the only ISO-standard in the area that must be taken into consideration. Still there are other 'standards' which are even stronger than RM-ODP. However, many of these other de facto standards are built on RM-ODP or are of the same origin. RM-ODP supports the view on the distributed system from five viewpoints (Table 6). Every viewpoint represents the system in a complete form, but they are targeted towards a particular audience. RM-ODP also defines a number of functions that are necessary to support a comprehensive RM-ODP platform. This can be considered to support the presented suggestion of the dimensions of functions group in the distributed system. RM-ODP also suggests the arrangement of the binding object (Figure 6) to hide the distributed environment, which is unstable by nature, from the application programs, which must be made stable.

At the same time, OOA has been completed and many candidates of system objects and classes especially with RM-ODP have been found. This is the natural cause of the fact that RM-ODP is a reference standard for distributed systems. The model is criticised of being too complex, but still it saved us from 'invent the wheel again' phenomenon. The results of the OOA will be deployed in the design phase in Chapter 5.

Furthermore, the State-of-the-practice section has been dealt with. Its aim was to find out the mini-concepts and the user requirements, not to compare the presented techniques. This is according to the established tactic. The results were carefully tabled to be used in the concept design in the next chapter. Many of the examined things belong to the OOA analysis and they will be used when the solution is to be realised in the later chapters. Mini-concepts were equally covered. The user requirements in Table 2 and Table 13 and the mini-concepts in Table 14 will be used in the next chapter as the data of the conceptual planning of the system. As the development of the system progresses over the design phase, the results will be checked against (and iterated over) this phase.

3. Conceptual planning

In this chapter, the conceptual solution is to be defined. As shown in the Road Map, QFD-method is to be applied [3]. The QFD-method is a quality management technique that translates the needs of the user into the technical requirements for a product. QFD concentrates on maximising user satisfaction. To accomplish this, QFD emphasises the understanding of what is valuable to the user and subsequently, the deploying of these values throughout the engineering process. When the method is applied, it does not matter how the information about user requirements, and on the other hand, how the technical requirements, both needed for the QFD-analysis, are reached – what matters is that the information is available when the QFD-matrix is initiated.

The work for reaching the necessary information was described in the previous chapters and the results of the user requirements are presented in Table 2 and Table 13. Applying the QFD-method to the problem starts with preparing the tables. First, user requirements from Table 1 and Table 13 will be combined together, then they are put into the QFD/A1-matrix with the mini-concepts from Table 14. Using only the A1-matrix to carry out the QFD-analysis is the most common way to deploy the method, but there are extended possibilities available with the method as well. However, what is intended is to support the understandability of the solution. The QFD-method will be used to isolate the best concepts, and by doing so, the problem is being generalised further. The result of analysing the QFD-matrix is a concept of a product, and hereby represents the deepest generalised level of an abstraction. The results of the analysis (the concept) will be used in the next chapter when the solution will be formed by presenting the architectural model of the system.

3.1 QFD-analysis

Because of too large tables, user requirements must be compressed. The compression can be done only by decreasing or removing redundancy among the requirements. Requirements in Table 2 and Table 13 are compressed as follows: PE1 overrides Tr1, PE4 overrides Tr14 and Tr15, and PE8 overrides Tr3. The result table is included in the QFD-matrix in Figure 4. The compression marks are included as well.

🟨X Mapping of the resource management associated features

▨X Mapping of the distribution associated features

M-c = Mini-concept
PE = Preferred Excitement
Tr = Treshold

Program logic is processed only on client-side	M-c 1	3																				
Program logic is processed only on server-side	M-c 2	3	3																			
Services join communities represented by the directory ..	M-c 3			1																		
Directory services contain more directory services ..	M-c 4			1	1																	
Directory service has a capability of containing simple ..	M-c 5			1	1	1																
Directory service has a capability of containing client ..	M-c 6						2															
Directory service has a capability of containing a descr..	M-c 7						2	2														
Server-side program logic and client-side program ..	M-c 8	3	3						3													
Thin or super-thin client with event and user screen ..	M-c 9	3	3						3	3												
Movable description of client-side program logic	M-c10						2	2			2											
Movable object contains client-side program logic ..	M-c11						2	2			2	2										
Program logic can be defined to run on client-side or ..	M-c12	3	3						3	3			3									
Services form into communities	M-c13			1	1	1								1								
Client-side program logic run-time condition is to be ..	M-c14	3	3						3	3			3		3							
Client-side program logic is secure against modification	M-c15															4						
Strategies to maintain resources can be set by manage..	M-c16	3	3						3	3			3				3					
Application program logic can be programmed ..	M-c17	3	3						3	3			3				3	3				
The distribution transparency is arranged by binding ..	M-c18	3	3						3	3			3				3	3	3			
Different capabilities of terminal devices are classified ..	M-c19	3	3						3	3			3				3	3	3	3		
The flexible channel structure enables the communication	M-c20	3	3						3	3			3				3	3	3	3	3	
The proxy object represents the application in the dist..	M-c21						2	2			2	2										2

		Mini-concept																				
User Requirements	Reference label	M-c 1	M-c 2	M-c 3	M-c 4	M-c 5	M-c 6	M-c 7	M-c 8	M-c 9	M-c10	M-c11	M-c12	M-c13	M-c14	M-c15	M-c16	M-c17	M-c18	M-c19	M-c20	M-c21
End-user can easily reach and deploy a software produ..	PE1			X	X	X						▨X		X	🟨X							
End-user gets his/her/its personal profiles for the equi..	PE2																🟨X					
End-user gets his/her/its personal profiles for a service	PE3																🟨X					
End-user has data and access security maintained ..	PE4															X	🟨X					
End-user will be provided a vision of the information ..	PE5																🟨X					
Software products or information services are consid..	PE6																🟨X					
Device provider can provide compatible terminal equip..	PE7																🟨X					
Service provider can provide rich applications types ..	PE8																🟨X					▨X
Service provider can provide software products or ..	PE9																🟨X					
Service provider can do business by trading software ..	PE10			X	X	X					▨X	▨X										▨X
Content provider can maintain the authorising and ..	PE11															X	🟨X					
The economy and convenience in sharing hardware res..	Tr2			X	X	X	▨X	▨X			▨X	▨X		X	🟨X							▨X
Easy API for application programmers	Tr4								🟨X									🟨X	🟨X		🟨X	
Adapt existing investments for software and systems	Tr5																	🟨X				▨X
Adapt existing applications	Tr6																	🟨X				▨X
Adapt existing operating systems	Tr7																	🟨X				
The ability to accommodate short-term changes ..	Tr8		🟨X																			▨X
The ability to accommodate medium-to-long terms ..	Tr9		🟨X																			
Managing and supporting users in a timely and cost..	Tr10		🟨X																			▨X
Ensuring exceptional application performances	Tr11	🟨X	🟨X						🟨X	🟨X			🟨X									
A fast and consistent response to the end-user's intera..	Tr12	🟨X											🟨X									
A measure of how small is the likelihood of the system ..	Tr13																🟨X					
Concept																						

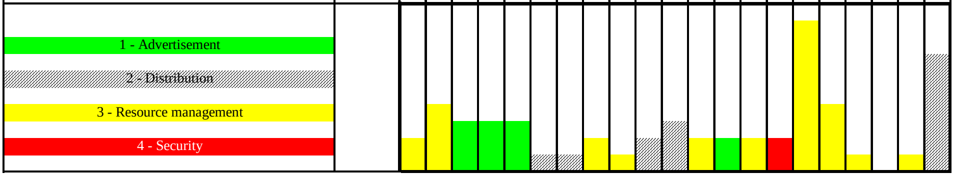


Figure 4. The QFD-matrix.

3.2 The concept

The concept can be taken from Figure 4 by analysing the QFD-matrix. In this section that analysis will be provided, and in the later subsections the findings will be investigated further. Individual strong features of the concept can be seen having many hits on the cross-reference field of the matrix (the field where the rows of user requirement and columns of the Mini-concept cross each other. These features are as follows:

- do not decide upon the client-server model until the very last moment, maybe as late as at run time (M-c 1, M-c 2, M-c12),

- divide the system into two layers - one to provide the support for applications, the other to provide support for maintenance (M-c17, M-c18),

- let the client (possibly called 'proxy') represent the application, so the client can be built to be as fat or as thin as necessary - and make it movable. A client should be able to contain the run time condition attributes of the information of the place where it was initiated for the first time (M-c11, M-c21),

- support user and device profiles to maintain the preferred settings of the end-user (M-c16),

- support information for management policy with many different levels of abstractions to maintain different viewpoints on distributed software (M-c16),

- support the system of local- or distributed mode, apply the distribution transparencies to hide the distributed mode from applications (M-c16, M-c17, M-c21),

- do not require management support as a mandatory feature, let application contain the information of whether full or partial management is needed in every particular case (M-c17),

- support the old application mode (e.g. EXE-mode) by a voluntary extension of management information to be added in the case the old application modes would like to get management support (M-c16),
- isolate the dimensions introduced in Table 2,
- define the concept of resources, how they should be managed,
- define the concept of service, how they should be understood, and
- consider the functions provided by a working application as a service for other entities of human or artificial origin.

3.2.1 Client-server model

It is justified to make the architecture of the client-server model [44, 45] as flexible as possible. Any particular model should not be chosen, rather the model should be able to choose differently depending on the application type and prevailing conditions of the execution environment in every individual case of the application use. The choice of model should aim at making the application's performance as good as possible. It should be possible to balance the model when programming, negotiating about its delivery, and even at run time when run time conditions change. As a summary: we must support client-server models of an ultra-thin or thin client, and fat client - each of them with multiple clients, and furthermore, the model with the ability to be adjusted between these models.

3.2.2 Layered structure

Dividing the system into two layers: one to support application modes and the other to support management features of the system is justified. This was assumed in Table 1 for the first time. Providing for the layer to be embedded in the underlying platform system, and on the other hand, the layer for support application modes to be run on the platform, should be an optional feature. The result of this optional choice is called compatibility; existing applications can continue their existence, but if the advanced application according to the new concept presented here is planned to be used, a corresponding support must be added to the underlying operating system. This is arguable by an example: if the application contains information for maintaining its security (as it should be according to the new concept), it must be made absolutely impossible to start the application without any support for the management of the security. Otherwise, there is no security.

3.2.3 Application with representatives

Applications should not be distributed as such. The economic choice is to distribute some part of an application, which can represent the application for users. This application model is used at least with the Jini-concept, and this proxy-mode can be equally adopted as such. However, it must be remembered that old applications models must be supported as well (e.g. EXE-mode), so the using of the represented mode of the application (the proxy mode) must be made an optional choice.

3.2.4 Profiles for user and devices

Profiles for the user’s personally preferred settings of configuration must be supported. Different settings are necessary when the user uses, for example, the same application with different terminal devices at different times. That means that the profile must have branches for the used applications as well as for the used devices, and have the user's preferred configuration for both.

3.2.5 Policies

Policies are the heart of the system. The policy of an application must contain all information about how the application is expecting to be treated in every particular execution environment and in the prevailing run time conditions. The policy could consist of a hierarchy of different levels of definition. The policy could be built to support different levels of the perception of the system, for instance, reflecting the viewpoint defined in RM-ODP (see Table 6). This may support the understanding and maintaining of the system between stakeholders. The issue of the policy driven management has been studied but not realised very far among the distributed systems. A further study of this issue will be left for the extension of this thesis. However, a few sources are referred here: [46, 47, 48, 49].

3.2.6 Distribution transparency

Difficulties that will exist as a consequence of the unstable nature of distributed execution environment should be hidden away from applications as well as from the management. This can be achieved by setting the responsibility of handling those difficulties to the special entities made for it. This kind of entity is called a Binding object in RM-ODP.

3.2.7 Burden by request

If some particular application type does not need maintenance at all (except for its policy that must always be maintained), or needs support just for some limited features, then only the relevant maintenance processes should be started. On the other hand, there may exist some execution platforms, which contain support only for a limited subset of maintenance features. In that case, an application should be able to work under the limited conditions as defined by its policy information. An application should not be able to work if it is against the term of the defined policy of the application.

3.2.8 Support for existing applications

Existing applications will be supported if they can express the information to create the policy about how every particular application is expected to be treated. This means that an application must contain its policy information. If an application does not contain any policy, it can run as it did before. It should be as follows: if an application contains information about policies, it must not be able to start at all on the environment that does not support the policy management feature.

3.2.9 Dimensions and their orthogonal position

In the correlation field of mini-concepts in Figure 4 (above the title Mini-concept), we can see the correlating mini-concepts as they are analysed. Regions of correlating concepts are separated by the numbers 1., 2., 3., and 4. These regions may be labelled by the features they represent: 1. advertisement, 2. distribution, 3. resource management, and 4. security. This is according to the assumptions made in Table 1.

3.2.10 Resource

It is necessary to define the meaning of the concept of resource the way it is considered. According to the sources [16], a resource is an abstraction. It is something that can be deployed by allocating (reserving) some based on the needs. In theory, resource signifies unlimited supply, as for example, the air we are breathing. In practice, there is not any unlimited supply of resources that software uses and therefore resources must be reserved and managed by the clear rules. Reservation of a resource may be done, e.g. when an application allocates some memory. There must be some entity that works as a gatekeeper for a resource. The gatekeeper must know how much of the particular resource has been reserved already, and whether the new reservation can be carried out anymore. There are many references in literature of these kinds of gatekeeper arrangements, for example [2] and [50]. The resource may be managed more or less adequately. Management of the resources reserved successfully by an application must be an optional feature to maintain the compatibility of the systems. Management cannot be required as a regular feature, so it should be made an optional choice. For keeping things simple, in the illustrations of the architecture of the system, the resource has been presented as an instantiation of a resource - an already successfully reserved share of some resource supply - viz. "Resource Instance". An extended presentation of the resources - how they really exist in software and hardware systems - we will have in Chapter 7.

3.2.11 Service

Similarly, to the definition of resource, the definition of service needs to be made clear to be used here. A service should be understood to be something that the entity called server provides. For example, we may say; "I am going to the dentist's to get some treatment for my teeth". Service is an abstract definition the way resource is. A service can be used only by using some instantiation of it.

3.2.12 Support for different component models

The model to be developed should be able to play the role of different component models. 'Successful large systems are combined from small successful systems', so application must be possible to be combined partially from different foreign components and equally, our model must be able to be part of foreign applications.

3.3 Summary

In this chapter, the concept of the system was defined. A conceptual solution represents the most generalised level of the problem, and the results represent the reusability of tacit human knowledge. In the entry, there were Table 2 and Table 13 concerning the user requirements, and Table 14 concerning the mini-concepts. The tables were mapped into QFD/A1-matrix in order to analyse them. The result of the analysis was presented as textual descriptions. Defining the conceptual solution from the QFD-table should not be considered as a piece of routine job - it is a creative process that deploys all analyst experiences and tacit knowledge from the problem domain - and thereby adds productivity and clearness on the creating process. The thinking process becomes transparent, repeatable and traceable, which are important features for making the definition of the concept more understandable. Presenting the thinking process in the table also supports the maturity levels of capabilities of the work process [51]. The concept will be elaborated on in the next chapter by designing the architecture based on the concept.

4. Architectural design

In this chapter, the concept that was defined in the previous chapter is to be realised. Consequently, the issue "how we will do it" will be discussed. As presented in the Road Map, an architectural solution is a realisation of a conceptual solution. Because there are many possibilities to implement the realisation from the same concepts, many justifications will be included during the processing for maintaining an understanding of the choices that have been made here.

Realisation of the concept is presented by means of the UML's [52, 53] abstract classes, binary associations and interface definitions. Models and notations recommended by UML be used as a minimum way, to make clear what is meant, not to give an impression of a clever analyst. The recommendations of UML encourage the use of different models, choosing only as many and as large as is necessary to make things understandable. On the other hand, the presented solution is intended to be a basis of the design model, so the model will be accurate enough after iterations.

The architectural model presents how the entities of the system are connected, and how they are interfacing with the world outside [54]. The result will be used in Chapter 5 as a basis of the design model in as an open-ended way as the style of UML provides for.

4.1 The client-server model

In Figure 5, the client-server schema - the balanced approach - used with the system is presented. The client of the server component represents the server's interface logic to its user. The application of the system is seen as a server who is serving its clients. The application must always be represented by some object that has a capability to intermediate between the application and its user concerning the application functionality (program logic). The user can be a human or an artificial entity, perhaps a group of people or another application.

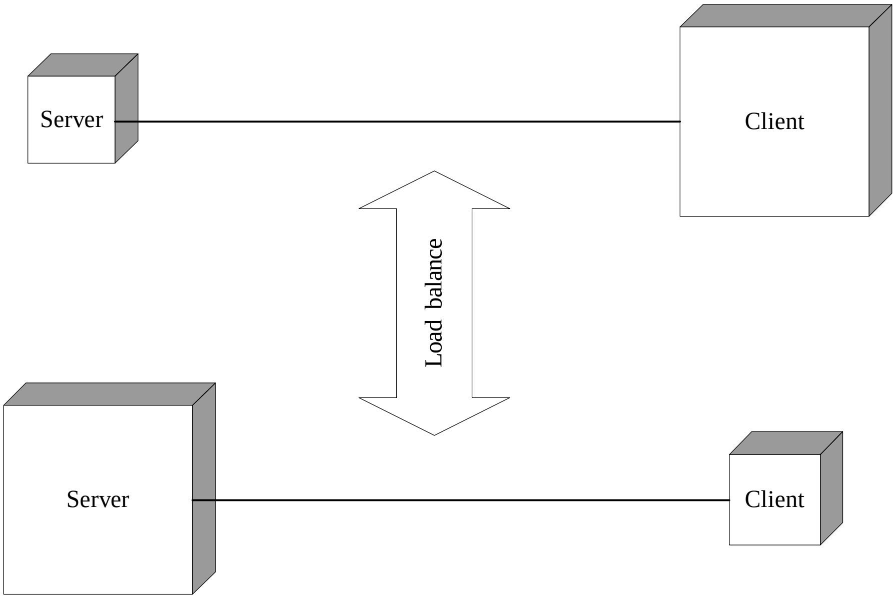


Figure 5. Load balancing.

Allocation of the computational processing needed to run the application (as a server) and its representative object as a client (later called proxy) totally on the client side (as is the case in the fat-client client-server model), or on the server side (as is the case in the server-based computing model), depends on the application type and the run time environments that exist in every particular case when the application is used. Examples include an application running on the platform as in a mobile phone, or the platform as a very powerful desktop computer or a network server. In other words, it should be possible to balance the client-server model in many phases: in the programming time, when loading to run, and even dynamically when run time conditions change while the program is running.

4.2 Resource

Resources and how they are managed, are two different things. An application program reserves (or allocates) an instance of some resource type for its use. The reservations do not depend on whether the instances of resources are managed or not. In a computer system, the resources must be guarded in some way by the gatekeeper, as told in Subsection 3.2.10. The same feature must be provided by the resource management system to be developed here, for any reservation of resource (resource instance) that an application makes. Manageability of a resource must be an optional feature - a programmer can make a choice whether he uses the resources in managed or unmanaged mode. This is because not all resources can be managed properly, at least at first. For example, Java class libraries cannot support management before they are modified. On the other hand, some applications may not need to be managed by including all their objects. Applications today exist mostly without management features. The choice must be made by the programmers of an application during the programming time.

The optional choice of using manageable resource instance can be arranged by interfaces. The interfaces may or may not be implemented by the programmers. The interface for the manageability of resource reservation (allocation) may be called "Manageable". More interfaces for a resource may be pointed out later. A couple of good candidates worth mentioning are "Movable" and "Cacheable".

4.3 Distribution transparency

Extra difficulties that occur because of the unstable nature of distributed environments can be hidden from the application programming and deploying by the distribution transparency properties of the system.

The distribution transparencies also make it possible for the systems that speak different languages, and run on different platforms, to communicate with each other in an understandable way. The transparencies also make it possible to deploy middle-ware techniques with the solution. Those techniques build a bridge between the systems that do not know each other.

The distribution transparencies will be arranged by deploying binding objects as illustrated in Figure 6. The binding objects must provide as many distribution transparency properties as is wanted or is necessary. Some of the possible transparencies are presented in Table 7, but more might be defined if there is any need for them, as mentioned in RM-ODP. At least in the networked environment, network transparency must be provided. Network transparency contains access and location transparencies (see Table 7 or [16]). These transparencies are necessary because the computational objects in the distributed environment must be able to access each other regardless of their location in the network. The computational object does not have to know where the other is exactly located. However, every computational object must be presented by a binding object by some unique name (UIN) or unique identifier (UID) extended with addressing information for the distributed environment. These will be used in the distributed environment to access computational objects. In addition to that the binding object must be capable of setting the address of the computational object, it must be capable of resolving the address of the computational object. The computational object must implement only the UIN name and/or UID, because these identifiers are needed only for distinguishing the computational object.

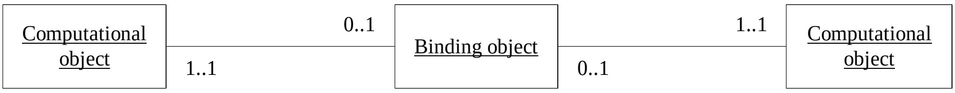


Figure 6. The Binding object.

4.4 Dimensions

The dimensions of the software object were introduced in Subsection 3.2.9 in Chapter 3. Four dimensions were defined, but more are possible to exist. Support for the dimensions can be arranged by defining the respective software object to maintain each dimension. Maintaining a separate dimension should be an optional feature. A correspondent process to support a separate dimension is to be created only if it is required by the policy of an application (policies were introduced in Subsection 3.2.5 in Chapter 3). To support a separate dimension in an application program, a programmer has to implement only the corresponding interfaces. The names of the interfaces are congruent with the correspondent dimensions, and they will be presented in Table 15 in Section 4.7. The interfaces also appear in the architecture models in Figure 12 and Figure 13.

4.4.1 Advertisement

The Advertisement dimension can maintain all the features that are necessary to make the existence of a service and its availability known for the interested user or for the public. Advertisement is maintained according to the corresponding advertisement policy of an application.

4.4.2 Distribution

The Distribution dimension can maintain all the features which are necessary to distribute or deliver everything that is needed to use the corresponding service in a compatible terminal or in whatever compatible device the user may have. Distribution is maintained according to the corresponding distribution policy of an application.

4.4.3 Resource management

The Resource management dimension takes care of the management of the instances of resources reserved by an application. Management follows the resource management policy that is defined by the application.

Resources are something that the application layer reserves or needs in other ways for its use. Application reserves and uses reserved resource instances not depending on whether the resources are managed or not. The Representative object and the Binding object must be defined as a resource as well. Instead, an application is something that wraps the objects and defines some logical behaviour of the system of software - this constitutes an application. Every reserved or created object is an instance of resource. When an application becomes an object (after initialising it), it becomes a service - because it can offer some useful function for others in some interiors. The services are represented by a proxy and can be used by other services or any entities.

A very simple application can exist only for making some of the resources available, shareable and therefore usable for others in the distributed environment, e.g. a printer becomes a printing service.

4.4.4 Security

The Security dimension can maintain all the features that are necessary to make the use of service secure for the interested user. Security is maintained according to the corresponding security policy of an application.

4.5 Policies

As mentioned in the concept definition in the previous chapter, policies are the heart of the system. Policies will exist on two levels: on the level of the description, and on the level of the object into which the information from the description is realised for the working computer system. There must be a way to express the information needed to create a policy object from the informal description. At this phase of the development process, the way to do this will not be defined, but one good candidate is XML. The policy may contain information from different administration domains. An assembled example can be presented: with a banking service it is necessary to maintain the policy information of the service not just by the bank, but by the bank's customer, who uses the service. These administration domains may need to overlap (the example ends).

Here are definitions for the policies:

- Advertisement policy - defines how to maintain the Advertisement dimension of a service.

- Distribution policy - defines how to maintain the Distribution dimension of a service.

- Resource management policy - defines how to maintain the Resource management dimension of a service.

- Security policy - defines how to maintain the Security dimension of a service.

4.6 Management model

The commonly used management model is presented in Figure 7. All Management decisions are based on the available Management information and the defined Management policy. Management actions are committed to maintain the "Manageable" resource instance. To reach the Management information and to apply the Management action by the corresponding Management policy, the Manageable and Policy interfaces must be known.

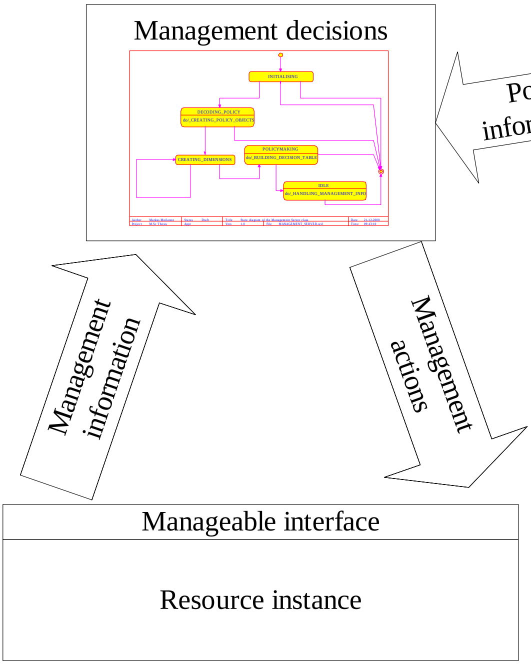


Figure 7. The common management model.

4.7 Layers

By dividing the system into two layers, application layer and maintenance layer, application can be isolated from the difficulties that are a consequence of an unstable distributed execution environment itself. To support compatibility, a demand to include the layers in the existing application modes and operating systems must be made optional. In Table 15, all combinations that can exist are presented.

Table 15. Combinations (Cb.) of management support.

Cb.	Application with Policy					
Management layer	Is a	no demands	Advertiseable	Distributable	Resourceable	Secureable
	Management	Works stand-alone	Works stand-alone *)	Works stand-alone *)	Works stand-alone *)	Works stand-alone *)
	Advertisement	Works stand-alone	Applic. is advertised according to its policy	Works in undistributed mode *)	Works with unmanaged resources *)	Works in unsecured mode *)
	Distribution	Works stand-alone	Works in unadvertised mode *)	Applic. is distributed according to its policy	Works with unmanaged resources *)	Works in unsecured mode *)
	Resource Management	Works stand-alone	Works in unadvertised mode *)	Works in undistributed mode *)	Applic. is managed by its resources according to its policy	Works in unsecured mode *)
	Security	Works stand-alone	Works in unadvertised mode*)	Works in undistributed mode *)	Works with unmanaged resources *)	Applic. will be secured according to its policy

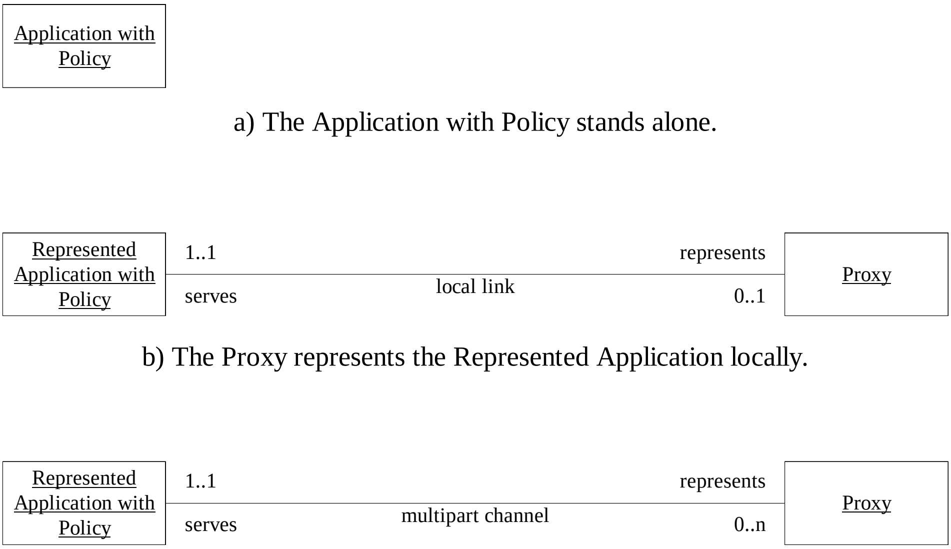
*) *if allowed by the associated dimension policy branch of an application*

The architectural model supports the free choice of combination and still respects the demand for every dimension set by the policy of an application. The burden that the system must carry will be minimised.

The need to fit up each layer with the support for some of the defined dimension must made optional too, as already mentioned in Section 4.4, and the choice must depend on the case of application type. Extra burden for execution environments and applications by supporting unnecessary dimensions when they are not really needed must not be forced to be added. However, applications should be able to work even if there is no management support present for all the dimensions in the platform, but only if working in such a way is allowed by the policy of this particular application.

4.7.1 Application layer and APIs

In Figure 8, the supported application modes are presented. Figure 8 a) shows the simplest mode available. The application stands alone still containing its policy information. Figure 8 b) shows the application mode that is locally represented by a proxy. The proxy knows how to interact with a user, and knows the run time condition releasing the user from making any configuration settings before starting the application. Figure 8 c) shows the application mode that can be represented by multiple proxies at the same time. The corresponding application logic to support multiple clients must be taken care of by the application's programmer.



a) The Application with Policy stands alone.

b) The Proxy represents the Represented Application locally.

c) Several proxies may be served by the Represented Application simultaneously in the distributed environment.

Figure 8. Supported application modes.

Application layer is defined by representing the interfaces (API's) of 1) the support for being with a policy, 2) the ability to be represented in the environment, and 3) the support for the Dimensions.

Interfaces for being with a policy are:

- "withPolicy" - an application explicitly defines its policy.

Interfaces for the ability to be represented are:

- "Representable" - an application can deliver a part that represents its functions for the application's user.

- "Proxyable" - an interface for Proxy-like deployment of objects in the environment.

Interfaces for the Dimension support are:

- "Advertiseable" - an application holds the information to maintain its advertisement in the environment.

- "Distributable" - an application holds the information to maintain the distribution of its representative part to the users of the application.

- "Resourceable" - an application holds the information to maintain the instance of resources that the application has been once allocated.

- "Securable" - an application holds the information about how its security must be maintained.

Interfaces will be seen in the architecture schemata later in Figure 12 and Figure 13 in Section 4.8.

4.7.2 Management layer

Figure 9 shows how a management server is serving an application. The application has a policy by which the management server can do the application's maintenance.

Figure 10 shows how a management server manages an application and its representative proxy locally. There is no distributed environment present.

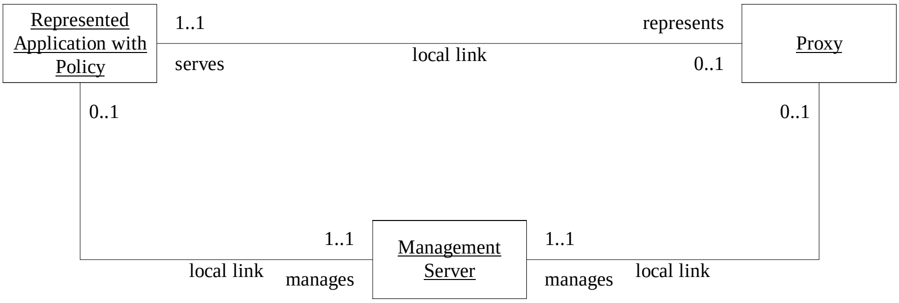


Figure 9. Management Server manages "Application with Policy".

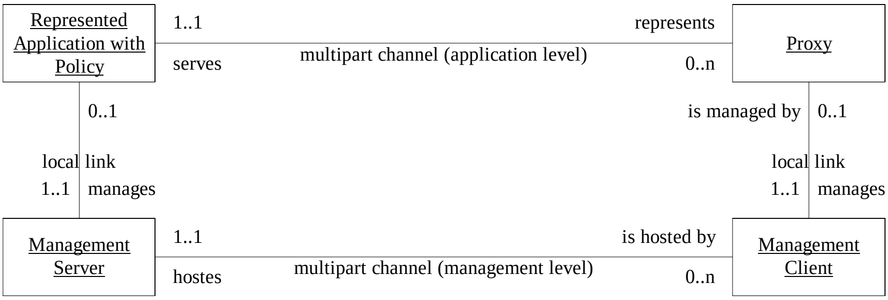


Figure 10. Management Server manages the "RepresentedApplication" and its Proxy using local links.

Figure 11 shows how a management server manages an application represented with multiple proxies in a distributed environment.

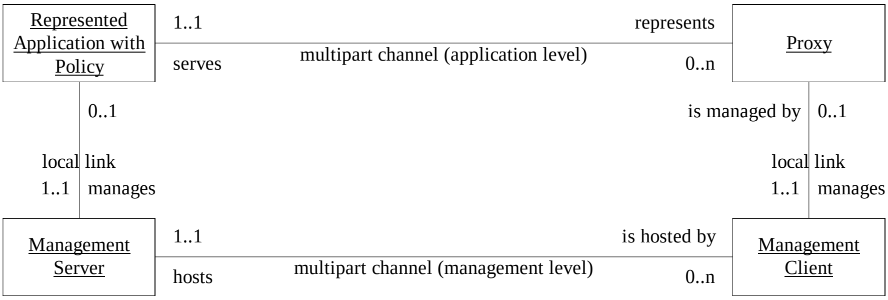


Figure 11. Management Server and its Clients manage the "RepresentedApplication" and its Proxies using multipart (communication) channels in the distributed environment.

The management supports the distributed proxies, but does not take care of the application program logic when it has to serve its multiple clients.

Management layer is defined by representing the interfaces (API's) of 1) the support for the deployment of the policy containing applications, on the different platforms, 2) the support for maintaining the Dimensions, 3) the support for the resource deployment, and 4) the support for the Distribution transparencies.

Interfaces for the deploying of the policy containing applications are:

- "Policy" - a template to build different policy-objects.

- "Management" - a server process has an ability to start an application and build the policy-object according to the information that the application holds.

- "Terminal" - a terminal object can start a new management server component with the application that the management component will further maintain. The terminal component can also maintain a group of created management components.

- "TerminalDevice" - a terminal device-object possesses resources and the terminal objects mentioned above.

Interfaces for maintaining the Dimensions are:

- "Dimension" - introduces the common properties of the dimensions, e.g. the ability to communicate.

- "Advertisement" - a management layer has the ability to maintain the Advertisement dimension by the given policy.

- "Distribution" - a management layer has the ability to maintain the Distribution dimension by the given policy.

- "ResourceManagement" - a management layer has the ability to maintain the Resource management dimension by the given policy.

- "Security" - a management layer has the ability to maintain the Security dimension by the given policy.

Interfaces for the resource deployment are:

- "Manageable" - resources can be made optionally manageable.

- "Movable" - a resource can be made movable (from one place to the other).

- Additional interfaces might be defined as the system develops further.

Interfaces for the distribution transparency are:

- "NetworkTransparency" - defines access and location transparencies in the distributed system.

- Additional distribution transparencies might be defined as the system develops further.

The presented interfaces will appear in the architecture schemata later in Figure 12 and Figure 13.

4.8 Architectural model

An architectural model is combined by connecting the parts that represent a concept [55, 56, 57, 58]. The parts are presented as abstracts. This is only to name them. When connecting the parts, corresponding relationships of entities, or objects, are considered. In addition to the architectural model illustrating how a system is combined internally, the model also illustrates how the system interacts with the rest of the world. The interaction points are presented with abstract interface definitions.

The behaviour of the system on this level of abstraction will be presented by means of the MSD (Message Sequence Diagram) in Subsection 4.8.3.

4.8.1 Architecture of the local system

In Figure 12, the Application has reserved Resources, which the Node (an individual computer system) possesses. The Management Server has to maintain the Resource instances following the Management Policy given by the Application. For this, the Resource Management Dimension Server, 'Resourcer', is to be created by the Management Server. After being created, the Resourcer will start to maintain the Resource management dimension of the Application.

The Resourcer is created by the Management Server only if the Application has any information that expresses the Resource Management Policy for the case.

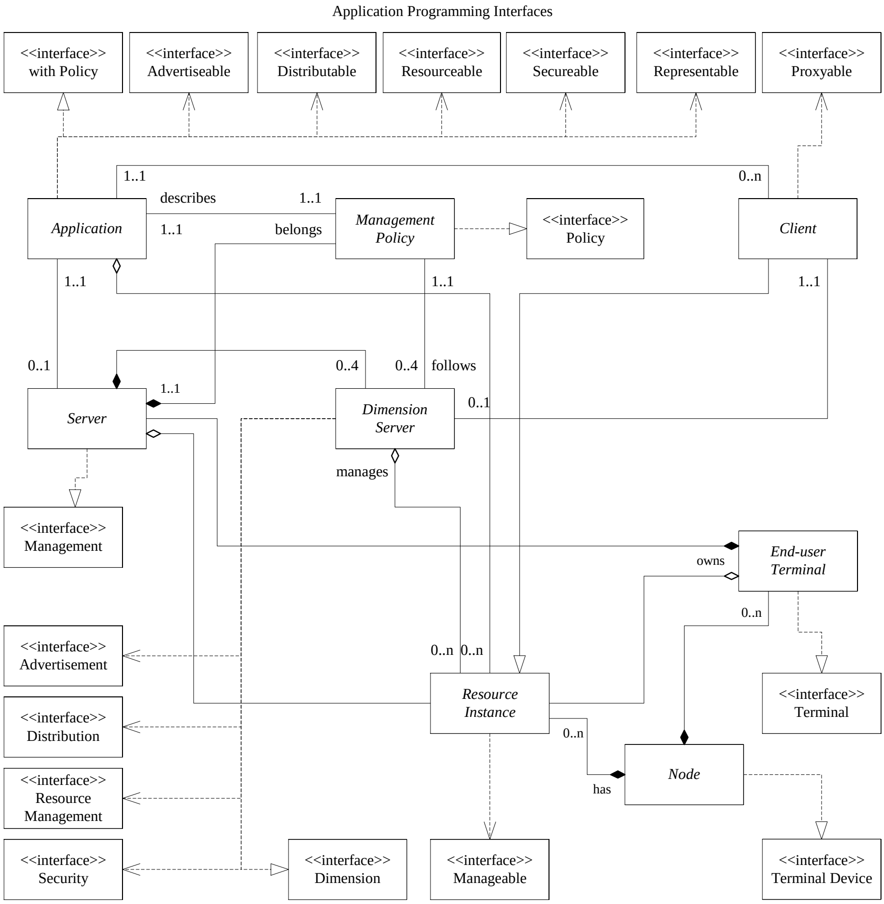


Figure 12. The architecture of the local system.

The Client is a resource and so it can be made optionally Manageable. The Client can be made into an optional supporter of Proxy-like deployment of an object by implementing the interface "Proxyable".

The Application must always implement the interface "withPolicy". If the implementation is not found by the Management Server, the Application cannot start at all with the support from the presented system. However, a common application can start as it was originally intended to do, but that remains outside the scope of this study.

An application can be made optionally resourceable by implementing the interface "Resourceable". When implementing the interface "Resourceable", a programmer is forced to write a proper management policy for that dimension as well. Other dimensions are handled the same way. A programmer of an application can support the management feature very easily just by defining a proper management policy.

4.8.2 Architecture of the distributed system

In Figure 13, we can observe the nature of the distributed system. Several concurrent clients may exist. Resources reserved from somewhere through the distributed environment do not exist. This is simply because those resources are served as services, already realised and maintained as shown by an example in Subsection 4.4.3. Services are usable by any Applications, one or many simultaneously, depending on the Management Policy of the Application. A programmer of an Application does not have to worry about programming support for the distribution, only define the proper management policy of how the management layer has to maintain the Application. However, even though the problems of the application management in the distributed environment are the responsibility of the management layer, the programmer should take care of the programming support for multi-user application mode itself when necessary. Fortunately, the well-managed Binding objects in question can be used for the communication. The problems usually exist with transactions, because multiple users' request may appear simultaneously.

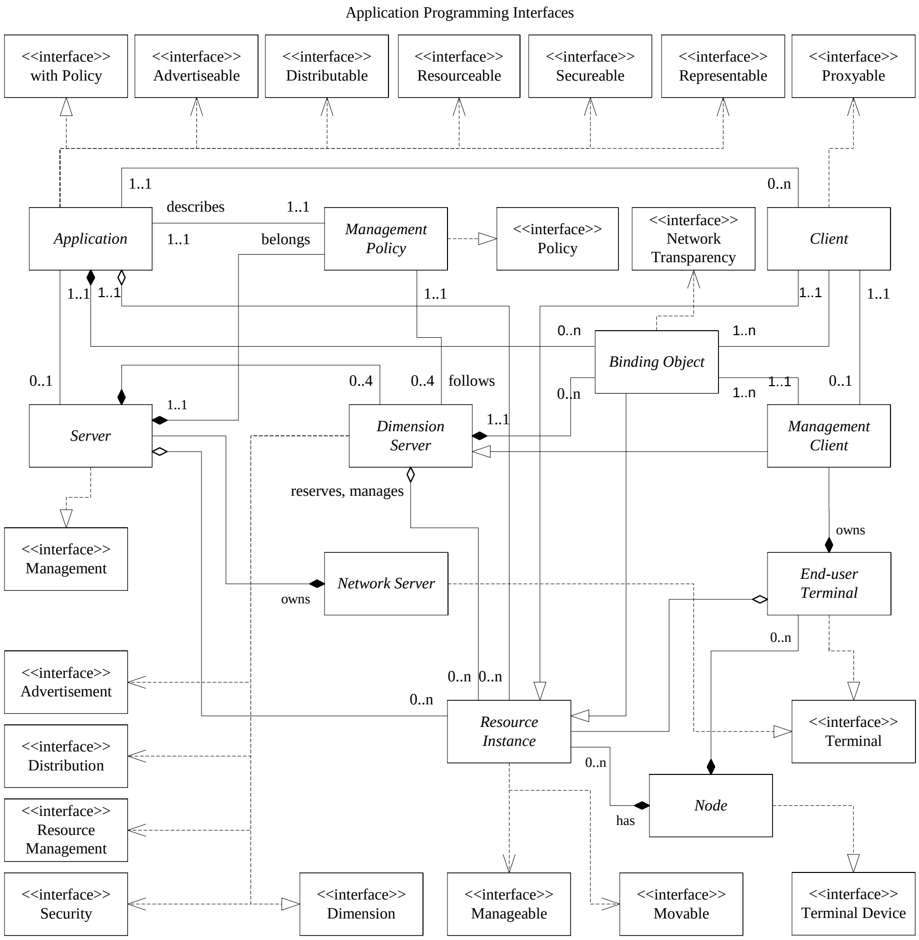


Figure 13. The architecture of the distributed system.

It must be made clear that a single computer node can be seen both as a distributed environment and as networked environment. The example includes the Microsoft Windows environment or Linux. There are many possibilities to deploy the distribution of the applications in such a way that has been presented here. Concerning the Windows environment, the technique of DDE (Dynamic Data Exchange), or the technique derived from it, e.g. OLE (Object Linking and Embedding) can be deployed to produce the access and location transparencies. Among other things, one useful service might be a printer share (see the example in Subsection 4.4.3).

Further, Figure 13 shows how architecture supports distribution transparency. Support is arranged by the Binding objects. These objects are software components, which implement a selected set of distribution transparencies. In the figure, an optionally selectable interface "NetworkTransparency" is shown, but many more can certainly be defined. The Binding object is a Resource Instance and so can be made optionally manageable.

The application layer is the same as it was with the local mode.

Differences between the local model (Figure 12) and the distributed model (Figure 13) are:

- The Management Client, which is hosted by the Dimension Server. The link needed for the client-server connection is arranged by the Binding object.

- Local links between the Application and its Client are replaced by the Binding objects.

- The Resource instance can be made movable (from one location to another) by implementing the interface "Movable".

- The system runs on two separate platforms, the Network Server and the End-user Terminal. Both of them implement the interface "Terminal".

4.8.3 Behaviour model of the architecture

Figure 14 illustrates the behaviour of the components in the architecture structures. An exact behaviour model cannot exist in this phase of the work. However, the presented MSD expresses quite well, how the system will act. This diagram presents only the case of the Distribution dimension being deployed.

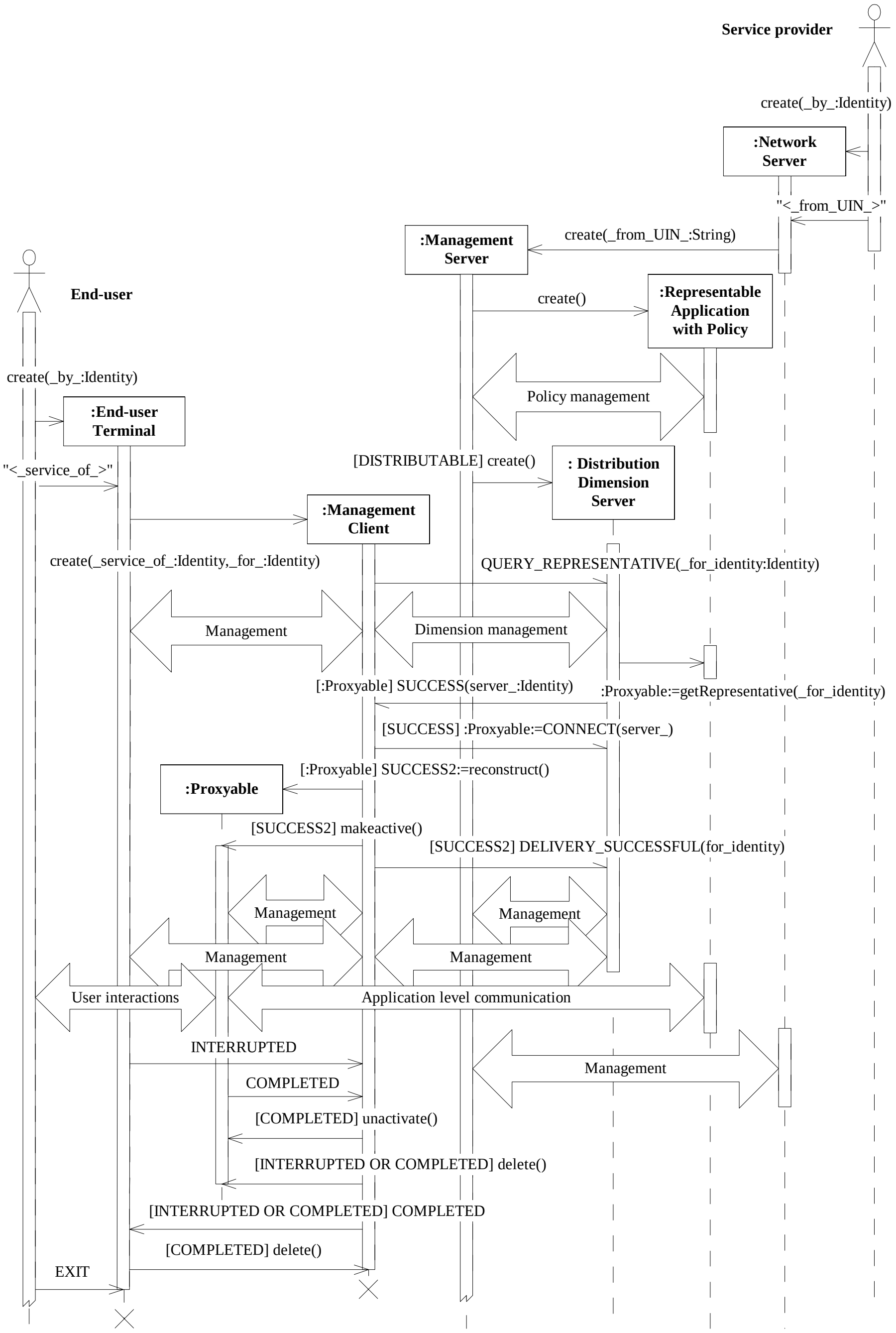

Service provider
create(_by_:Identity)
:Network Server
"<_from_UIN_>"
:Management Server
create(_from_UIN_:String)
End-user
create()
:Representable Application with Policy
Policy management
create(_by_:Identity)
:End-user Terminal
"<_service_of_>"
[DISTRIBUTABLE] create()
: Distribution Dimension Server
:Management Client
create(_service_of_:Identity,_for_:Identity)
QUERY_REPRESENTATIVE(_for_identity:Identity)
Management
Dimension management
[:Proxyable] SUCCESS(server_:Identity)
:Proxyable:=getRepresentative(_for_identity)
[SUCCESS] :Proxyable:=CONNECT(server_)
:Proxyable
[:Proxyable] SUCCESS2:=reconstruct()
[SUCCESS2] makeactive()
[SUCCESS2] DELIVERY_SUCCESSFUL(for_identity)
Management
Management
Management
Management
User interactions
Application level communication
INTERRUPTED
Management
COMPLETED
[COMPLETED] unactivate()
[INTERRUPTED OR COMPLETED] delete()
[INTERRUPTED OR COMPLETED] COMPLETED
[COMPLETED] delete()
EXIT

Figure 14. Behaviour model of the architecture when distributing the representative of the application.

4.9 Summary

In this chapter, the architectural design was conducted. The basis of the design was the conceptual model presented in Chapter 3. Two models were presented one for a local environment and other for a distributed environment as an extension of the local mode. It is worth mentioning that a single computer node can be seen as a local environment as well as a distributed environment (what the networked environment is by nature). For example, in the Microsoft Windows it is possible to deploy dimensions between applications the way that has been done by the technique derived from DDE.

The architectural models were presented by the abstractions of entities and abstractions of interfaces to make it possible for the system to be equipped with real functionality for every possible platform or operational system which may exist.

It became clear that many different architectural models could be implemented from the same conceptual model. The particular solution presented is a result of several iteration steps over the conceptual planning as well as the levels above it. In the first place, applicability of the architectural model has to be validated against the conceptual model. The produced architectural model isolates the programming support for the maintenance from the programming of the application program logic. This feature is provided by the interfaces, which can be implemented optionally.

The issue of how an application program reserves resources for its use was illustrated. Furthermore, there was also an illustration of how the reservation of the resources does not mean that the instances of resources will be managed. Management support for the instance of resource was arranged by defining corresponding interfaces, which was made optional. In addition, a presentation of how to support a system working in the distributed environment by defining an entity called a "Binding object" was provided. The Binding object provides the transparency of distribution. The functions of interfaces, classes and their hierarchy and other open questions will be defined in the next chapter when we enter the design phase of the system.

5. System design

The architecture of the system has been designed in the previous chapter; next, it will be realised as UML design models [59, 60]. The architecture was presented utilising abstractions and interfaces introduced by UML. Referring to the Road Map, the models in the Design phase reflect the Requirements analysis phase, and thereby the models must be evaluated against the results of the phase. We will turn to use the standards, design patterns and mini-concepts found during the Requirements analysis phase and which were evaluated afterwards in the Architectural design phase. To validate the Design model, we will pay attention to the Requirements Specification.

Because the open-ended models are being used, the architecture and the design model derived from it will match, at least after the iterations.

Due to the limited time and space resources of this work, the limitations of the work must be defined - not all the design models will be presented, only the abstraction of "Distributable and Representable Application with Policy", and the corresponding management layer to support it. However, this limitation will be compensated by presenting a full implementation of this application mode in the rest of the chapters. It is worth pointing out that even the limited solutions will be quite large.

The design model will be presented with great accuracy. Thereby the aim is at supporting the deployment of the automatic implementation tools (code generation) to function with the system in the future. Today, code generation tools are shipped with most of the UML software. Of course, the code generation tools can be used after the programming language and UML-tools specific adaptation of the models.

In the design phase, the static and dynamic models must be presented. The static model can be presented by utilising the UML Class Diagrams. To present the dynamic model, there are many possible diagram types to deploy with UML: Message Sequence Diagram, Collaboration Diagram, Activity Diagram, and State Chart Diagram. From these, the SCD (State Chart Diagram) has been chosen because of the aim at the deployment of the code generation feature of the UML tools. Commonly, by SCDs, the dynamic behaviour of classes or the dynamic behaviour of operations in classes can be presented.

In this case, the static model can be produced by expanding the interfaces of the architectural models. The behaviour model of the architecture was presented in Figure 14 in Chapter 4, and it gives us a manuscript of the dynamic of the distribution-related classes in the system.

5.1 Static models

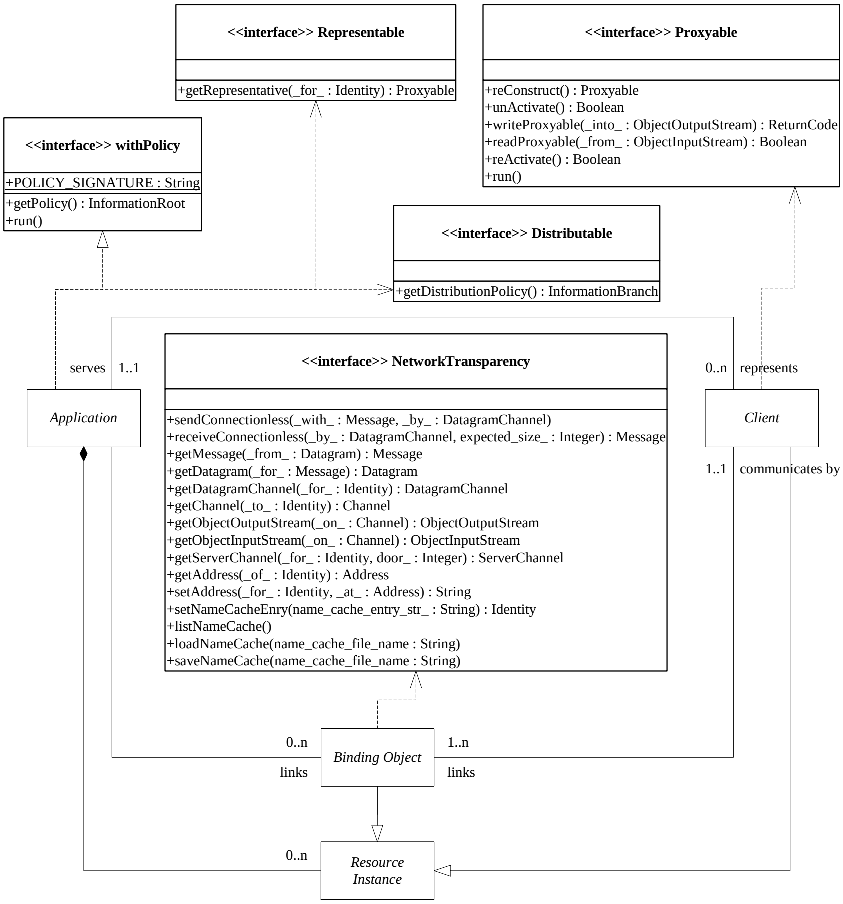
The static model of a Distributable Application and the corresponding management layer to support the distribution will be presented. Any special application will not be presented only the application model. The application-specific program logic must be programmed when the corresponding application is programmed. However, an example implementation of an application, the "HelloWorldApplication", will be presented in Appendix A.

The distribution of an application is provided by distributing the part that will represent the application in the environment. This part will be supported by the "Interface Proxyable". The distribution follows the application-specific policy and it is maintained by the management layer. The management layer is described by presenting exact static and dynamic models as long as the models can be presented without the environment, platform, and programming language-specific implementations.

5.1.1 Class diagram of a "Distributable" application

When any specific application program is produced, the presented abstract models must be implemented by implementing the interface functions and constants. An example of the implementation will be offered in Chapter 6, as already mentioned. In Figure 15, the necessary interfaces of a "Distributable" application are published.

The explanations and comments of the interface operations will be found in the source listing in Appendix A.



<<interface>> Representable
+getRepresentative(_for_ : Identity) : Proxyable
<<interface>> Proxyable
+reConstruct() : Proxyable
+unActivate() : Boolean
+writeProxyable(_into_ : ObjectOutputStream) : ReturnCode
+readProxyable(_from_ : ObjectInputStream) : Boolean
+reActivate() : Boolean
+run()
<<interface>> withPolicy
+POLICY_SIGNATURE : String
+getPolicy() : InformationRoot
+run()
<<interface>> Distributable
+getDistributionPolicy() : InformationBranch
<<interface>> NetworkTransparency
+sendConnectionless(_with_ : Message, _by_ : DatagramChannel)
+receiveConnectionless(_by_ : DatagramChannel, expected_size_ : Integer) : Message
+getMessage(_from_ : Datagram) : Message
+getDatagram(_for_ : Message) : Datagram
+getDatagramChannel(_for_ : Identity) : DatagramChannel
+getChannel(_to_ : Identity) : Channel
+getObjectOutputStream(_on_ : Channel) : ObjectOutputStream
+getObjectInputStream(_on_ : Channel) : ObjectInputStream
+getServerChannel(_for_ : Identity, door_ : Integer) : ServerChannel
+getAddress(_of_ : Identity) : Address
+setAddress(_for_ : Identity, _at_ : Address) : String
+setNameCacheEnry(name_cache_entry_str_ : String) : Identity
+listNameCache()
+loadNameCache(name_cache_file_name : String)
+saveNameCache(name_cache_file_name : String)
serves
1..1
Application
0..n
represents
Client
1..1
communicates by
0..n
links
Binding Object
1..n
links
0..n
Resource Instance

Figure 15. Class diagram of a "Distributable" application.

The diagram's ability to explain itself is trusted upon by the undersigned, but a few clarifications will be provided, nevertheless. The interface "Network Transparency" makes the "Binding Object" class provide the Location Transparency, because a computational object can locate another computational object only by being informed about the "Identity" of the other instead of the actual physical address. The Identity is expressed in the system by a unique name or/and unique identifier. The Network Transparency must have the ability to resolve the Identity, therefore when the public function of "getAddress (_of_: Identity): Address" will be implemented, the resolving feature must be taken care of. For that, local names cache or/and a public name directory service must be utilised. The ability of a computational object to inform of its identity to the name service is one of the responsibilities of the Advertisement dimension of the object. More information about the transparencies can be found in Section 4.3 and Table 7, and information about the Advertisement dimension can be found in Section 4.4. The Access Transparency feature is supported as well, because only the locally known data types can be used for the communication.

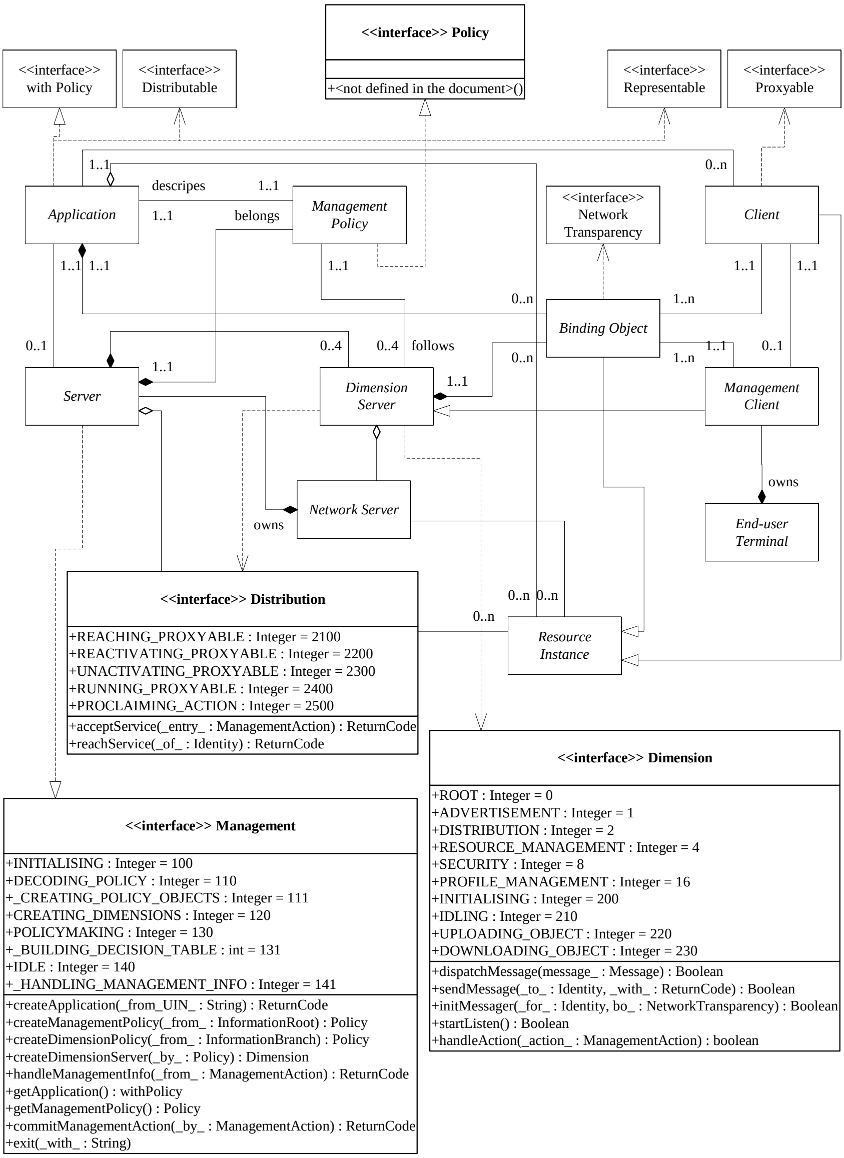
The interface "Proxyable" maintains the system's ability to handle the clients during its life cycle in the distributed environment. The interface "withPolicy" forces the static String attribute of "POLICY_SIGNATURE" to be included in an application in the programming time - so the application will be recognised and its policy will be maintained accordingly.

In the diagram, there are many new data types introduced which have not been included before. Examples include e.g. "ReturnCode" and "Datagram". These are utility classes of the system. How the utility classes will be implemented depends on the environment for which they were intended, but the system must know how they can be handled. The system's ability to handle them can be provided by introducing the interfaces that the utility classes must implement. If the utility types are transferred between the computer nodes in the distributed environment, the Access Transparency takes care of the type transformation to the local environment. The utility classes are introduced in Table 16.

The management layer exists only with the Application object. The components presenting the management layer will not be initialised without some application needing them to maintain its management. For every single application object, there is a set of management objects. However, for one application there may exist as many management client object sets as there are representatives of the application distributed.

Figure 16 illustrates the management layer that corresponds to the application mode presented in Figure 15. It is capable of distributing a "Proxyable" client of the application. The interfaces to support the management are "Management", "Policy", "Dimension", and "Distribution". The functions and constants to support the state machines to model the dynamic behaviour of their implementation will be presented.

The naming convention is perceivable, too. The usefulness of the naming convention used will emerge as the study continues with the SCDs.



<<interface>> Policy
+<not defined in the document>()
<<interface>> with Policy
<<interface>> Distributable
<<interface>> Representable
<<interface>> Proxyable
1..1
0..n
Application
descripes
1..1
1..1
belongs
Management Policy
<<interface>> Network Transparency
Client
1..1
1..1
1..1
1..1
1..1
0..n
1..n
Binding Object
0..1
0..4
0..4
follows
0..n
1..1
1..n
0..1
1..1
Server
Dimension Server
1..1
Management Client
owns
Network Server
owns
End-user Terminal
0..n
0..n
0..n
Resource Instance
<<interface>> Distribution
+REACHING_PROXYABLE : Integer = 2100
+REACTIVATING_PROXYABLE : Integer = 2200
+UNACTIVATING_PROXYABLE : Integer = 2300
+RUNNING_PROXYABLE : Integer = 2400
+PROCLAIMING_ACTION : Integer = 2500
+acceptService(_entry_ : ManagementAction) : ReturnCode
+reachService(_of_ : Identity) : ReturnCode
<<interface>> Dimension
+ROOT : Integer = 0
+ADVERTISEMENT : Integer = 1
+DISTRIBUTION : Integer = 2
+RESOURCE_MANAGEMENT : Integer = 4
+SECURITY : Integer = 8
+PROFILE_MANAGEMENT : Integer = 16
+INITIALISING : Integer = 200
+IDLING : Integer = 210
+UPLOADING_OBJECT : Integer = 220
+DOWNLOADING_OBJECT : Integer = 230
+dispatchMessage(message_ : Message) : Boolean
+sendMessage(_to_ : Identity, _with_ : ReturnCode) : Boolean
+initMessager(_for_ : Identity, bo_ : NetworkTransparency) : Boolean
+startListen() : Boolean
+handleAction(_action_ : ManagementAction) : boolean
<<interface>> Management
+INITIALISING : Integer = 100
+DECODING_POLICY : Integer = 110
+_CREATING_POLICY_OBJECTS : Integer = 111
+CREATING_DIMENSIONS : Integer = 120
+POLICYMAKING : Integer = 130
+_BUILDING_DECISION_TABLE : int = 131
+IDLE : Integer = 140
+_HANDLING_MANAGEMENT_INFO : Integer = 141
+createApplication(_from_UIN_ : String) : ReturnCode
+createManagementPolicy(_from_ : InformationRoot) : Policy
+createDimensionPolicy(_from_ : InformationBranch) : Policy
+createDimensionServer(_by_ : Policy) : Dimension
+handleManagementInfo(_from_ : ManagementAction) : ReturnCode
+getApplication() : withPolicy
+getManagementPolicy() : Policy
+commitManagementAction(_by_ : ManagementAction) : ReturnCode
+exit(_with_ : String)

Figure 16. Class diagram for the Distribution management layer.

All the interfaces introduced before (in Figure 15) as well as the interfaces introduced now, will stand, even the interfaces which will be introduced in the future, are final as well. For the sake of clarity, the introduced interfaces are listed here as an example: "NetworkTransparency", "withPolicy", "Distributable", "Representable", and "Proxyable". The explanations and comments of these interface operations will be found in the source listing in Appendix A.

The interfaces, which are now expanded in Figure 16, will be commented also on in the source code listing in Appendix A.

5.2 Dynamic models

The purpose of the dynamic models is to present the exact behaviour of the system objects when they interact with each other. In accordance with the chosen tactics in the beginning of this chapter, only the SCDs will be presented. It must be notified that the dynamics of the application (the program logic) is application-specific - this will not be discussed here, while the dynamics of the objects on the management layer must be considered. A set of management components will be left on the abstraction level; however, its dynamic behaviour can still be illustrated. This can be done by utilising the functions and the constants presented in the static models in Section 5.1. The model will be precise as long as any programming language or local environment depended notation will not have to be taken into consideration.

5.2.1 State diagrams of the Management Server class

Figure 18 illustrates an SCD for the class “Management Server”. SCD will be deployed when the abstract class "Server" is realised by implementing the interface "Management". The extended sub-state machines "_CREATING_POLICY_OBJECTS", "_BUILDING_DECISION_TABLE", and "_HANDLING_MANAGEMENT_INFO" will be presented by the code listings in Chapter 6 when the system is implemented. The mini-specifications of the utility classes are written in to Table 16. The implementation of the utility classes (interfaces) as well as the functions (methods) bodies will be presented in the implementation example in Chapter 6 as well.

Next, the basic principle of the main diagram of "MANAGEMENT_SERVER" will be explained. Thereby the information about the application intended to be started on the platform is passed as the string parameter of "_from_UIN". The string parameter represents the application UIN (i.e. the name of the application). The first operation of the management is to find out the preconditions before offering management support for the application. This will be done in the state "INITIALISING". If the application cannot be identified or even located at all, the state fires the Fail-condition and exits. If the application is identified to be one without the "POLICY_SIGNATURE", it does not concern the system presented here and may accordingly be passed over. The third case is that the application is identified as carrying the "POLICY_SIGNATURE", so its management must be taken care of. The instance of the application was created at this point. Next, the machine decodes the policy information that the application expresses. First in the state "DECODING_POLICY", the instance of the "Management Policy" object is created, then the "Dimension Policy" objects are created in the sub-state machine "_CREATING_POLICY_OBJECTS". If this fails, the machine exits, informing of an error. Created policy objects are inserted in the array "_on_demand". Next, the "Dimension Server" objects, which implement the corresponding dimension interfaces, will be created in the state "CREATING_DIMENSIONS". The objects are inserted in the array "_to_support". It must be notified that it is possible that the platform in question cannot offer support for any, or some, of the dimensions that the application policy demands. Whether the application can nevertheless be started will be decided in the state "POLICYMAKING". The decisions are then marked into the two-dimensional array of the "decision_table" in the sub-state "_BUILDING_DECISION_TABLE". The policy may allow the execution even with an insufficient support provided by the platform. However, the management will exit, if the support reflecting the application policy cannot be fulfilled. The "decision_table" contains the same information as Table 15 in Chapter 4.7. The table contains the "Decision" types. The type is expressed by the three truth-values of DEMANDS, SUPPORTED, and JUDGMENT (see Section 5.3) according to the situation in hand. The one false JUDGMENT will cause the system to exit, otherwise the machine will enter the state "IDLE" and start to handle the management information in the sub-state "_HANDLING_MANAGEMENT_INFO". The handling is mostly done by routing the information among the dimension servers.

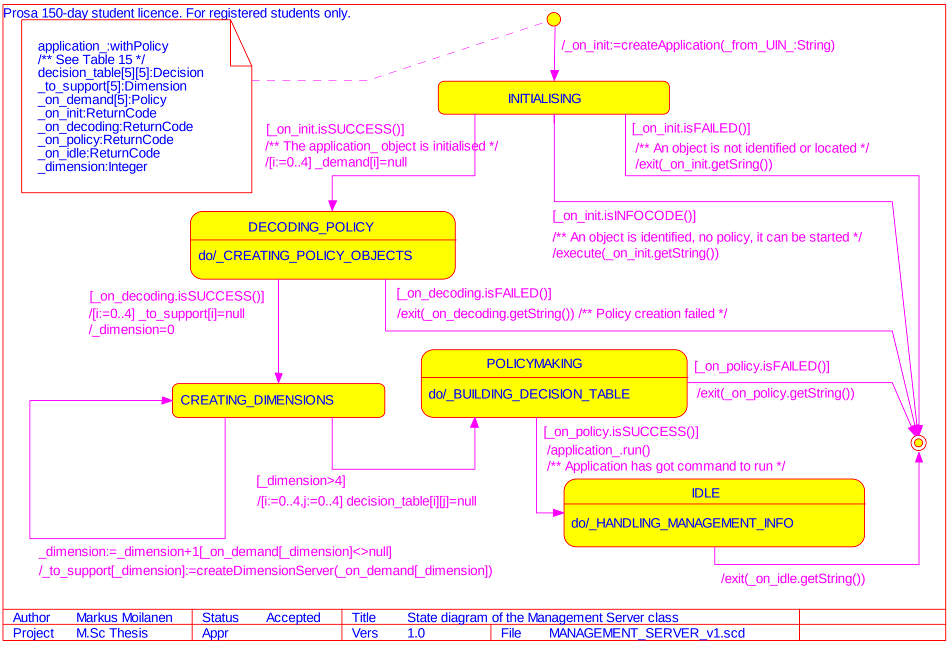

Prosa 150-day student licence. For registered students only.
application_:withPolicy
/** See Table 15 */
decision_table[5][5]:Decision
_to_support[5]:Dimension
_on_demand[5]:Policy
_on_init:ReturnCode
_on_decoding:ReturnCode
_on_policy:ReturnCode
_on_idle:ReturnCode
_dimension:Integer
/_on_init:=createApplication(_from_UIN_:String)
INITIALISING
[_on_init.isSUCCESS()]
/** The application_ object is initialised */
/[i:=0..4] _demand[i]=null
[_on_init.isFAILED()]
/** An object is not identified or located */
/exit(_on_init.getSring())
DECODING_POLICY
do/_CREATING_POLICY_OBJECTS
[_on_init.isINFOCODE()]
/** An object is identified, no policy, it can be started */
/execute(_on_init.getString())
[_on_decoding.isSUCCESS()]
/[i:=0..4] _to_support[i]=null
/_dimension=0
[_on_decoding.isFAILED()]
/exit(_on_decoding.getString()) /** Policy creation failed */
POLICYMAKING
do/_BUILDING_DECISION_TABLE
[_on_policy.isFAILED()]
/exit(_on_policy.getString())
CREATING_DIMENSIONS
[_on_policy.isSUCCESS()]
/application_.run()
/** Application has got command to run */
[_dimension>4]
/[i:=0..4,j:=0..4] decision_table[i][j]=null
IDLE
do/_HANDLING_MANAGEMENT_INFO
_dimension:=_dimension+1[_on_demand[_dimension]<>null]
/_to_support[_dimension]:=createDimensionServer(_on_demand[_dimension])
/exit(_on_idle.getString())
Author Markus Moilanen
Status Accepted
Title State diagram of the Management Server class
Project M.Sc Thesis
Appr
Vers 1.0
File MANAGEMENT_SERVER_v1.scd

Figure 17. State diagram of the Management Server class.

5.2.2 State machine of the Distribution Dimension Server class

The state machine in the Distribution Dimension Server takes care of transferring the representative of the application object into the End-user Terminal environment. The class Distribution Dimension Server is a realisation of the abstract class of Dimension Server. The interface "Dimension" must always be implemented by it. In this particular case, the class implements the interface "Distribution", but in many other cases it would implement more interfaces to support extra dimensions correspondingly. It is also possible to support the scalability feature of the system in providing concurrent (or parallel) Dimension Server instances.

There must be management client programs somewhere in the distributed environment that may ask for the proxyable. They may send the communication primitive "QUERY_REPRESENTATIVE", when they want to deploy the service presented by the application. When Distribution Dimension Server receives the primitive, it proclaims the action in the state of

"PROCLAIMING_ACTION" among the other dimensions, and when permission is granted, it will enter the "UPLOADING_OBJECT" state. The utility class ObjectUploader will do the actual job - it will do very much the same as any usual file transfer protocol is expected to do. These will be presented by the code listings in the Appendix A.

The system will not have to try to get messages through by repeating the transmissions - the Binding Object will take care of all the communication. The Binding Object that implements the "MessagingTransparency" as a sub set of the "NetworkTransparency", will notify the end-user when the transmission is found totally impossible - usually this may signify some fatal error in the network, e.g. the network is totally lost. Also, it needs to be notified that the Dimension Distribution Server does not see any difference between local and distributed modes. To access another computational object, it just calls this by its Identity. This is because of the Binding object that arranges the Distribution Transparency and is thus able to resolve the Identity.

5.2.3 State machine of the Management Client class

In the distributed environment, the Management Client acts as a client in the client-server system for the Distribution Dimension Server presented in Subsection 5.2.2. When the client is started, it will try to contact the server side and will ask for a proxyable by sending the message with the primitive “QUERY_REPRESENTATIVE”. Then it will wait for the answer. When the answer from the right Identity has been received, it will download a proxyable by getting connected to the given service access point. As soon as the load operation is committed, it will try to start the proxyable client program. The associated states of “REACHING_PROXYABLE” and “DOWNLOADING_OBJECT” will do the same job that the "PROCLAIMING_ACTION" and “UPLOADING_OBJECT” do - this was depicted in Subsection 5.2.2 - but representing, however, the client of the client-server system. The state machines will be presented by the source code listings in the Appendix A.

It must be notified that it is not guaranteed that there is the associated client code (Java class in the Java environment) present in the client environment. The proxyable represents only the living attributes of its associated client object, not its executable code. Consequently, the systems support the minimalist way of deploying applications. The situation is the same when the client program logic (represented by the proxy) is transferred by deploying the XML or even HTML scripts - they do not contain any target platform specific executable code, either. One must distinguish the actual executable code from the descriptions of interactions presented by the scripts mentioned. To be executed, the scripts must be interpreted by a web browser. The problem of providing the client side with the corresponding executable code resource is one of the duties that the Resource Management Dimension must take care of. This kind of lack of resources is congruent with the depicted situation that will exist with XML or HTML scripts if the web browser program is not available in the environment. This subject will be elaborated on in Chapter 7.

5.3 Utility classes

Classes that are deployed throughout the system are utility classes, and they will eventually be placed in their own utility package. The implementation depends on the target system to which the management layer is meant to be applied. The utility classes are explained in Table 16.

Table 16. Utility classes.

Class	Description
Address	Defines the address of an entity in the distributed environment. The address contains logical information to deliver mail into a correct location, and it is extended with the Identity information to pass the mail to the correct Identity. It is like an Internet-address, extended with the port and the Identity information.
Channel	Might be implemented as a TCP/IP's Socket in Java to present the end-point for the connection-oriented data transfer.
The table continues.	

Continuing Table 16.

Class	Description
Datagram	Might be implemented as a TCP/IP's connectionless message presented by the DatagramPacket in Java (see Table 4).
Datagram Channel	Might be implemented as the TCP/IP's DatagramSocket in Java for the connectionless message transfer.
Decision	Contains the truth-value types of the attributes "DEMANDS", "SUPPORTS", and "JUDGMENT", and includes the corresponding operations to handle the attributes.
Identity	Defines the identity of an entity. Identity must be unique and it may consist of the UIN part for the human friendly name, and the UID part for the exact identifier for the machine origin users.
Information Branch	A description from which the dimension associated policy-object will be created.
Information Root	A description from which the root policy -object will be created. The implementation of this will not be presented in this thesis.
Management Action	Will contain all the necessary information to handle the transaction in the systems.
Message	The logical information that the entities may exchange. The information is represented by a String type.
ReturnCode	The enumerated attribute type of "FAILED", "SUCCESS", "INFOCODE", and the String-type attribute.
Server Channel	Might be implemented as a TCP/IP's ServerSocket in Java to present the service access point of the communication system for the connection-oriented data transfer.

The utility classes in this phase of the product life cycle represent the base classes of their class hierarchies, and classes that will exist in the future of this product will descend from the base classes. However, classes will still work as they were planned to work due to the polymorphism feature of the object model deployment. On the other hand, any decision upon the implementations of the utility classes will not be made, but naturally, some implementations of them have to be written for the demos.

In addition to the presented utility classes that will become the interfaces, a few more exist in the models. These are "Boolean", "Integer", "String", "Object", "ObjectInputStream", and "ObjectOutputStream". These classes are so common among the programming platform that there is no reason to adapt them by some interface-definition in this phase. However, when the resource management actions must take place, these classes will have to be specialised.

5.4 Summary

The system design phase is now completed. Many large diagrams and drawings were produced with several iterations steps over the implementation (coding) phase. Generally, it seems that there is no simple way to model the whole program. To model the static existence of the system might be a little bit easier than to model the dynamic behaviour (the program logic) - but still, the modelling was worth striving for. This will be proved when the product continues its existence - there is the graphic, easy to maintain, and the model which presents the same system in every possible programming language, and simultaneously, for every possible environment. With correct tools, the necessary program code can be generated from the models for every purpose. This will denote an even bigger advancement when the model is put under the version control system - the version control can be extended to control the whole product. The SCM (Software Configuration Management) tool to control products can be integrated in the UML tool as well.

UML offers many alternate diagram types to model. Concerning the work focussed in this thesis, the CLDs have been chosen for the static models and the SCDs to model the dynamic behaviour of the classes. One good reason for this choice was the fact that most of the UML tools use only these two diagram types for the code generation.

The architecture model was realised as UML design models. Because the job of designing a full system seemed to be outside the time and space resources of this work, a decision upon the limitation of the job was made by deciding to realise only one dimension of the four - the Distribution dimension. However, this dimension was decided to be modelled to the end, including the implementation phase as well.

Furthermore, there were many utility classes involved. These classes were presented by means of the Mini-specifications. All the models and utility classes will be implemented in the next chapter utilising the Java programming language.

6. Implementation

In this chapter, the design models presented in the earlier chapter will be implemented. The Java -programming language will be used [61, 62, 63].

The goal is to clarify the solution presented by the design models so that it can easily be deployed when the solution is applied for different purposes in the future. The fact is that it is not appropriate to bind the solution with any special programming language. However, the Java programming language is commonly used today instead of PDL (Program Design Language), especially with OOD.

The solution type is a framework, not an endproduct. When the framework is applied to a special implementation of the management capability, a final program code will be created.

The Java programming language offers a way to present the abstract methods with the abstract classes. Thus the state machines could be modelled using this approach, but a preference is made to use the example application to clarify the functions of the state machines. Application programmers can then use the example application as a template for their own implementations.

Java's "interface" definition could be used to define a new reference data type [53]. The new reference type could be used as if it were an actual data type. The type can be referred to, but it does nothing without its implementation. This is quite appropriate for the purpose of this thesis - it is not necessary to define any actual data types, still the principles of the solution can be explicitly implemented. The same would have been provided by using the UML templates [6] in order to use templates to present the abstract classes and data types; nevertheless the former way was chosen for this work. The code will be produced manually to achieve the best possible structure of the code. In addition, the hierarchy of the applicable class library of software reflecting the UML Packages will be presented.

There is a slight problem in using the Java interface "Serializable" considering the distribution demo: serialising and de-serialising object serialises only the non-static attributes of the objects - not the associated byte code resource. Even this approach is convenient and commonly used by other languages, too; it will corrupt the demo presentation unless the byte code resource is transferred to the target environment before the actual object serialising. Normally, providing the target platform with resources will be the responsibility of the resource management, however, this feature included in the distribution service to maintain proxies for the demo, will be provided here. It is worth of mentioning here that Java RMI has the feature of transfer the byte code implemented as default. However, RMI is not deployed here, but it could be deployed in the future implementations of the presented solution.

In this chapter, only the essential parts and all the necessary descriptions of the example application will be presented; the full program code is listed in the Appendix A.

6.1 Packages

Figure 18 shows the package hierarchy of the system. Package "fi.vtt.ele.management_framework.implementation" contains the interface definitions of the design models presented earlier in their corresponding sub-packages. In the implementation phase, package structures will be realised in a form of class library.

In the figure, package "fi.vtt.ele.management_framework.implementation. utility" is browsed open and associated names of classes are shown as an example. It appears that there are three unmentioned classes presented: Messager, ObjectDownloader and ObjectUploader. The classes will be commented in Table 19. They represent classes of the kind which it is possibly to build on the interfaces of the class library - even though they are actual classes, they can be compiled with the library without any realisation of the interfaces. The way we can provide the generic state machines in a platform independent way - in the future, this is an important feature of the solution. Applications can be simulated before the actual realisation of the interfaces, and the realisations can be built simply for emulating the real world objects.

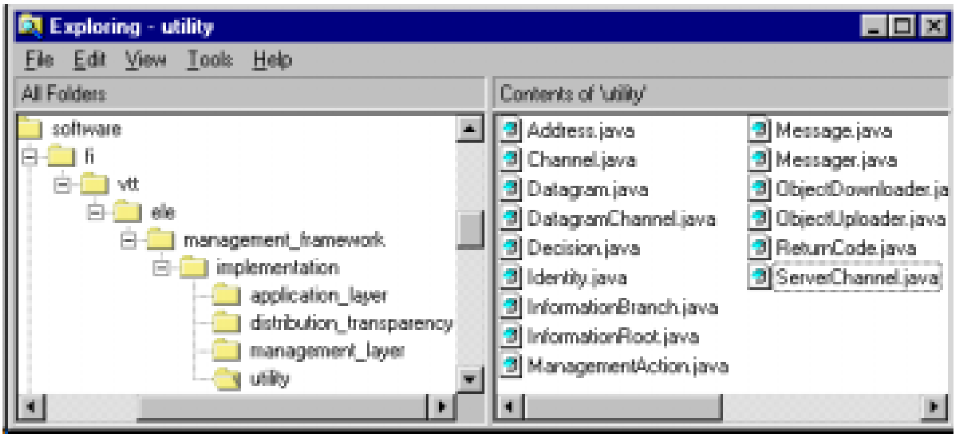


Figure 18. Package hierarchy of the Java classes of the system.

6.2 Interfaces

Interface definitions of the system are presented in this section. The Java implementation of the interfaces is presented. The associated name of the packages to compare to the class hierarchy illustrated in Figure 18 can be read after the word "package" on the right column of Table 17. The source code listing of the distribution demo in the Appendix A will not include the code presented here - the demo program only uses this library. Thus, the demo reflects the way the library will usually be deployed in practice. Table 17 introduces the interfaces used on the application layer and it reflects the diagram in Figure 16. The NetworkTransparency will also be used by the management layer. The library is designed to be implemented also in using other programming languages, such as C++ and Assembly. To realise the solution, the interfaces must be implemented in a platform dependent way. As already mentioned, an example implementation will be presented. Table 18 introduces the interfaces used on the management layer and it reflects the diagram in Figure 16; and finally, Table 19 introduces the interfaces in the utility package and it reflects the definitions in Table 16. The whole library package can be compiled independently from the platform - this works everywhere as long as the Java is present. The package is commented further in the source code listings in the Appendix A.

The library is designed to be implemented also in using other programming languages, such as C++ and Assembly. To realise the solution, the interfaces must be implemented in a platform dependent way. As already mentioned, an example implementation will be presented.

Table 17. Interfaces of a "Distributable" application.

Interface	Java implementation

withPolicy	package fi.vtt.ele.management_framework.implementation
	.application_layer;
	import
	fi.vtt.ele.management_framework.implementation.utility.*;
	public interface withPolicy
	{
	  public final static String
	POLICY_SIGNATURE = "WITH_POLICY";
	  public InformationRoot getPolicy();
	  public void run();
	}

Distributable	package fi.vtt.ele.management_framework.implementation
	.application_layer;
	import
	fi.vtt.ele.management_framework.implementation.utility.*;
	public interface Distributable
	{ public InformationBranch getDistributionPolicy(); }

Representable	package fi.vtt.ele.management_framework.implementation
	.application_layer;
	import
	fi.vtt.ele.management_framework.implementation.utility.*;
	public interface Representable
	{
	  public Proxyable getRepresentative(Identity _for_);
	}

Proxyable	package fi.vtt.ele.management_framework.implementation
	.application_layer;
	import fi.vtt.ele.management_framework.implementation
	.utility.ReturnCode;
	import java.io.ObjectOutputStream;
	import java.io.ObjectInputStream;
	public interface Proxyable
	{
	  public Proxyable reConstruct();
	  public boolean unActivate();
	  public boolean writeProxyable(ObjectOutputStream into_);
	  public boolean readProxyable(ObjectInputStream from_);
	  public boolean reActivate();
	  public void run();
	}

The table continues.	

<table>
<tr><th colspan="2"><em>Continuing Table 17.</em></th></tr>
<tr><th>Interface</th><th>Java implementation</th></tr>
<tr><td>Network Transparency</td><td><pre>
package fi.vtt.ele.management_framework.implementation.
distribution_transparency;
import
fi.vtt.ele.management_framework.implementation.utility.*;
import java.io.ObjectOutputStream;
import java.io.ObjectInputStream;
public interface NetworkTransparency
{
  public void sendConnectionless
                  (Message _with_, DatagramChannel by_);
  public Message receiveConnectionless
             (DatagramChannel _by_, int expected_size_);
  public Message getMessage(Datagram _from_);
  public Datagram getDatagram(Message _for_);
  public DatagramChannel getDatagramChannel
                                        (Identity _for_);
  public Channel getChannel(Identity _to_);
  public ObjectOutputStream getObjectOutputStream
                                          (Channel _on_);
  public ObjectInputStream getObjectInputStream
                                          (Channel _on_);
  public ServerChannel getServerChannel
                             (Identity _for_, int door_);
  public Address getAddress(Identity _of_);
  public String setAddress(Identity _for_, Address _at_);
  public Identity setNameCacheEntry
                          (String name_cache_entry_str);
  public void listNameCache();
  public void loadNameCache(String name_cache_file_name_);
  public void saveNameCache(String name_cache_file_name_);
}
</pre></td></tr>
</table>

Table 18. Interfaces of the management layer with the "Distribution" support.

<table>
<tr><th>Interface</th><th>Java implementation</th></tr>
<tr><td>Policy</td><td><pre>
/** This interface and its definitions is created only for
the test purpose. This is the place in where the policy
must be interpreted from XML-based descriptions of
InformationBranch and Information Root. */
package
fi.vtt.ele.management_framework.implementation.management.l
ayer; import
fi.vtt.ele.management_framework.implementation.utility.*;
public interface Policy extends InformationBranch
{ /*The interface InformationBranch will be implemented
                                        for the demo.*/ }
</pre></td></tr>
<tr><td colspan="2"><em>The table continues.</em></td></tr>
</table>

Continuing Table 18.

Interface	Java implementation

Dimension	package
	fi.vtt.ele.management_framework.implementation.management_l
	ayer;
	import fi.vtt.ele.management_framework.implementation.
	            distribution_transparency.NetworkTransparency;
	import
	fi.vtt.ele.management_framework.implementation.utility.*;
	public interface Dimension
	{ public final static int ROOT = 0;
	  public final static int ADVERTISEMENT = 1;
	  public final static int DISTRIBUTION = 2;
	  public final static int RESOURCE_MANAGEMENT = 4;
	  public final static int SECURITY = 8;
	  public final static int PROFILE_MANAGEMENT = 16;
	  public final static int INITIALISING = 200;
	  public final static int IDLING = 300;
	  public final static int UPLOADING_OBJECT = 400;
	  public final static int DOWNLOADING_OBJECT = 500;
	  public boolean dispatchMessage(Message message_);
	  public boolean sendMessage(Identity _to_, ReturnCode
	with);
	  public boolean initMessager(Identity _for_,
	                                NetworkTransparency bo_);
	  public boolean startListen();
	  public boolean handleAction(ManagementAction _action_);
	}

Distribution	package
	fi.vtt.ele.management_framework.implementation.management_l
	ayer;
	import
	fi.vtt.ele.management_framework.implementation.utility.*;
	public interface Distribution
	{ public final static int REACHING_PROXYABLE = 2100;
	  public final static int REACTIVATING_PROXYABLE = 2200;
	  public final static int UNACTIVATING_PROXYABLE = 2300;
	  public final static int RUNNING_PROXYABLE = 2400;
	  public final static int PROCLAIMING_ACTION = 2500;
	  public ReturnCode acceptService(ManagementAction
	entry);
	  public ReturnCode reachService(Identity _of_);
	}

Management
 	
(continues)
	package fi.vtt.ele.management_framework.implementation
	                                                  .management_layer;
	import fi.vtt.ele.management_framework.implementation
	                                                         .utility.*;
	import fi.vtt.ele.management_framework.implementation
	                                      .application_layer.withPolicy;

The table continues.

Continuing Table 18.

Interface	Java implementation

Management *(continuing)*	public interface Management
	{
	  public final static int INITIALISING = 100;
	  public final static int DECODING_POLICY = 110;
	  public final static int _CREATING_POLICY_OBJECTS = 111;
	  public final static int CREATING_DIMENSIONS = 120;
	  public final static int POLICYMAKING = 130;
	  public final static int _BUILDING_DECISION_TABLE = 131;
	  public final static int IDLE = 140;
	  public final static int _HANDLING_MANAGEMENT_INFO = 141;
	  public ReturnCode createApplication(String _from_UIN_);
	  public Policy createManagementPolicy(InformationRoot _from_);
	  public Dimension createDimensionServer(Dimension _for_, Policy _by_);
	  public ReturnCode handleManagementInfo(ManagementAction _from_);
	  public ReturnCode commitManagementAction
	                 (Identity _on_, ManagementAction _with_);
	  public withPolicy getApplication();
	  public Policy getManagementPolicy();
	  public void exit(String _with_);
	}

Table 19. Interfaces and classes in the utility package.

Interface	Java implementation

Address	package fi.vtt.ele.management_framework.implementation.utility;
	public interface Address
	{ public void setHost(String host_str_);
	  public String getHost();
	  public void setDoor(int door_);
	  public int getDoor();
	  public String to_String();
	}

Channel	package fi.vtt.ele.management_framework.implementation.utility;
	import java.io.InputStream;
	import java.io.OutputStream;
	public interface Channel
	{ public InputStream getInputStream() throws Exception;
	  public OutputStream getOutputStream() throws Exception;
	  public void close();
	}

The table continues.

Continuing Table 19.

Interface	Java implementation

Datagram	package
	fi.vtt.ele.management_framework.implementation.utility;
	public interface Datagram
	{
	  public String getContents();
	  public void setContents(String contents_);
	  public void setToAddress(Address to_address_);
	  public void setFromAddress(Address from_address_);
	  public Address getToAddress();
	  public Address getFromAddress();
	  public String to_String();
	}

Datagram Channel	package
	fi.vtt.ele.management_framework.implementation.utility;
	public interface DatagramChannel
	{
	  public void sendDatagram(Datagram datagram_)
	                                          throws Exception;
	  public Datagram receiveDatagram
	                  (int expected_size_) throws Exception;
	  public void setTimeout(int timeout_) throws Exception;
	  public int getDoor();
	}

Decision	package
	fi.vtt.ele.management_framework.implementation.utility;
	public interface Decision
	{
	  public boolean DEMANDS();
	  public boolean SUPPORTS();
	  public boolean JUDGMENT();
	}

Identity	package
	fi.vtt.ele.management_framework.implementation.utility;
	public interface Identity
	{
	  public void from_String(String identity_str_);
	  public String to_String();
	}

Information Branch *(continues)*	/** This interface and its definitions is created only for the test purpose. This is the place in where the descriptions of the policy must be developed as further study - might be XML-based. */ package fi.vtt.ele.management_framework.implementation.utility;

The table continues.

Continuing Table 19.

Interface	Java implementation

Information Branch *(continuing)*	public interface InformationBranch
	{ /** The next definition is only for the demo - definitions will be separated in the next version. */
	public int getDIMENSION();
	/** The next definitions are ROOT-related - they will be separated in the next version. */
	public String getUIN();
	public String getUID();
	/** The next definitions are DISTRIBUTION-related - they will be separated in the next version. */
	public int getPREFERRED_DOOR();
	public int getMAX_NUMBER_OF_REPRESENTATIVE_OUT();
	}

Information Root	package fi.vtt.ele.management_framework.implementation.utility;
	public interface InformationRoot
	{
	/** Must be implemented. See Table 16 and the previous item.*/
	}

Management Action	package fi.vtt.ele.management_framework.implementation.utility;
	public interface ManagementAction extends Message
	{
	public Message createMessage();
	public boolean adaptMessage(Message _message_);
	public int getIntvalue();
	public long getLongvalue();
	public Object getObject();
	public void setIntvalue(int intvalue_);
	public void setLongvalue(long longvalue_);
	public void setObject(Object object_);
	}

Message	package fi.vtt.ele.management_framework.implementation.utility;
	public interface Message extends ReturnCode
	{
	public Identity getFromIdentity();
	public Identity getToIdentity();
	public void setToIdentity(Identity to_identity_);
	public void setFromIdentity(Identity from_identity_);
	public ReturnCode createReturnCode();
	public boolean swapIdentities();
	public boolean adaptReturnCode(ReturnCode returncode_);
	}

The table continues.

Continuing Table 19

<table>
<tr><th>Interface</th><th>Java implementation</th></tr>
<tr><td>Messager

<em>(continues)</em></td><td><pre>
package
fi.vtt.ele.management_framework.implementation.utility;
import fi.vtt.ele.management_framework.implementation.

distribution_transparency.NetworkTransparency;
import
fi.vtt.ele.management_framework.implementation.management_l
ayer;
/** Utility class Messager is utility enabling
communication between Identities. It can be installed as a
thread or as usual instance. If installed as a thread, it
receives messages in a loop and will call the listener
introduced by the parameter listener. Listener must
dispatch the message to the correct Identity, or if the
message cannot be interpreted, return false.*/
public class Messager implements Runnable
{
  private Identity for_id = null;
  private NetworkTransparency bo = null;
  private Dimension listener = null;
  private DatagramChannel datagram_channel= null;
  private int receive_buffer_size = 512;
/**
  @param for_id_ will own the communication channel.
  @param bo_ is a binding object implementing
                                the network transparency.
  @param listener_ is the object which implements
 the interface "Dimension". It must dispatch the messages.
*/
  public Messager(Identity for_id_, NetworkTransparency
              bo_, Dimension listener_) throws Exception
  {
    for_id = for_id_; bo = bo_; listener = listener_;
    datagram_channel = bo_.getDatagramChannel(for_id);
  }
  public synchronized void sendMessage
                        (Message _with_) throws Exception
  {
    bo.sendConnectionless(_with_, datagram_channel);
  }
  public void setTIMEOUT(int timeout_)
  {
    try
    {
      datagram_channel.setTimeout(timeout_);
    } catch (Exception e) {/* do not care */}
  }
  public Message receiveMessage()
  {
</pre></td></tr>
</table>

The table continues.

Continuing Table 19.

<table>
<tr><th>Interface</th><th>Java implementation</th></tr>
<tr><td>Messager
(continuing)</td><td>

```
    try
    {
        Message message = bo.receiveConnectionless
                 (datagram_channel, receive_buffer_size);
      if (listener != null)
                    {listener.dispatchMessage(message);}
      return message;
    } catch (Exception e) { return null; }
  }
  public void run() {while (true) receiveMessage();}
}
```

</td></tr>
<tr><td>Object
Downloader
(continues)</td><td>

```
package
fi.vtt.ele.management_framework.implementation.utility;
import fi.vtt.ele.management_framework.implementation.
                                     management_layer.*;
import fi.vtt.ele.management_framework.implementation.
           distribution_transparency.NetworkTransparency;
import java.io.ObjectInputStream;
/**
  Utility class ObjectDownloader provides the system
capability
  of download objects. It is the other end of the object
transfer of
  the system.
*/
public class ObjectDownloader implements Runnable
{
  private ManagementAction entry = null;
  private Dimension dimension = null;
  private ObjectInputStream objectstream = null;
  private Channel channel = null;
/**
  @param entry is a record of a client action to take care.
  @param bo_ is a binding object implementing
                                  the network transparency.
  @param dimension is an object which is able
                                 to handle the return code.
*/
  public ObjectDownloader(ManagementAction entry_,
     NetworkTransparency bo_, Dimension dimension_) throws
Exception
  {
    entry = entry_;
    dimension = dimension_;
    channel =  bo_.getChannel(entry_.getFromIdentity());
    objectstream = bo_.getObjectInputStream(channel);
  }
```

</td></tr>
</table>

The table continues.

Continuing Table 19.

<table>
<tr><th>Interface</th><th>Java implementation</th></tr>
<tr><td>Object Downloader <i>(continuing)</i></td><td><pre>
/** downloadObject() -methods is called for do the actual
                                       work of download */
 public void downloadObject()
  {
    try
    {
      Object o = objectstream.readObject();
      entry.setObject(o);
      objectstream.close();
      channel.close();
      entry.setSUCCESS();
// Inform the dimension object about a successful download
and wait until it has handled the object.
    if (dimension != null) {if
                   dimension.handleAction(entry)) return;}
    } catch (Exception e)
    {
      entry.setFAILED();
      if (dimension != null)
                   {dimension.handleAction(entry); return;}
    }
  }
  /**The Runnable method*/
  public void run() { downloadObject(); }
}
</pre></td></tr>
<tr><td>Object Uploader

<i>(continues)</i></td><td><pre>
package
fi.vtt.ele.management_framework.implementation.utility;
import fi.vtt.ele.management_framework.implementation.
          distribution_transparency.NetworkTransparency;
import fi.vtt.ele.management_framework.implementation.
                                       management_layer.*;
import java.io.ObjectOutputStream;
/**
  Utility class ObjectUploader provides the system
  capability of delivering the necessory objects between
  the identities with its other end of ObjectDownloader.
  It is implementation independent and therefore
  compileable with its associated interface Dimension.
*/
public class ObjectUploader implements Runnable
{ private ManagementAction entry = null;
  private ServerChannel serverchannel = null;
  private Dimension dimension = null;
  private NetworkTransparency bo = null;
  /**
    @param entry_ is a client action which must be handled.
</pre></td></tr>
</table>

The table continues

<table>
<tr><th colspan="2"><em>Continuing Table 19.</em></th></tr>
<tr><th>Interface</th><th>Java implementation</th></tr>
<tr><td>Object Uploader <em>(continuing)</em></td><td><pre>
  @param bo_ is a binding object implementing
                                  the network transparency.
  @param dimension is the object which must handle
                                       the return code. */
  public ObjectUploader(ManagementAction entry_,
         NetworkTransparency bo_, Dimension dimension_)
                                          throws Exception
  {
    entry = entry_; bo = bo_; dimension = dimension_;
    serverchannel = bo_.getServerChannel
            (entry_.getToIdentity(), entry_.getIntvalue());
  }
  /**Method uploadObject carries out the sending of
                                                the data */
  public void uploadObject()
  {
    try
    { /** Waiting for the connect from
                                  the ObjectDownloader. */
      Channel channel = serverchannel.accept();
      ObjectOutputStream objectstream =
                        bo.getObjectOutputStream(channel);
      objectstream.writeObject(entry.getObject());
      objectstream.flush();
      objectstream.close();
      entry.setSUCCESS();
   /**It is about time to inform the dimension about
                                           the work done.*/
    if (dimension != null) {dimension.handleAction(entry);}
    } catch (Exception e)
    { entry.setFAILED();
      if (dimension != null)
                         dimension.handleAction(entry);}
    }
  }
  /**The Runnable method*/
  public void run() {uploadObject();}
}
</pre></td></tr>
<tr><td>Server Channel</td><td><pre>
package
fi.vtt.ele.management_framework.implementation.utility;
public interface ServerChannel
{
  public Channel accept() throws Exception;
  public int getLocalDoor() throws Exception;
  public Address getLocalAddress() throws Exception;
  public void close() throws Exception;
}
</pre></td></tr>
<tr><td colspan="2"><em>The table continues.</em></td></tr>
</table>

Continuing Table 19.

Interface	Java implementation

ReturnCode	package
	fi.vtt.ele.management_framework.implementation.utility;
	public interface ReturnCode
	{ public final static String query = "QUERY";
	  public final static String maximum = "MAXIMUM";
	  public final static String busy = "BUSY";
	  public final static String granted = "GRANTED";
	  public final static String denied = "DENIED";
	  public final static
	               String representative_proxy =
	"REPRESENTATIVE_PROXY";
	  public final static String class_byte_code_resource
	                           = "CLASS_BYTE_CODE_RESOURCE";
	  public final static String failed = "FAILED";
	  public final static String success = "SUCCESS";
	  public final static String infocode = "INFOCODE";
	  public boolean isQUERY();
	  public boolean isMAXIMUM();
	  public boolean isBUSY();
	  public boolean isGRANTED();
	  public boolean isDENIED();
	  public void setQUERY();
	  public void setMAXIMUM();
	  public void setBUSY();
	  public void setGRANTED();
	  public void setDENIED();
	  public boolean isREPRESENTATIVE_PROXY();
	  public void setREPRESENTATIVE_PROXY();
	  public boolean isCLASS_BYTE_CODE_RESOURCE();
	  public void setCLASS_BYTE_CODE_RESOURCE();
	  public boolean isSUCCESS();
	  public boolean isFAILED();
	  public boolean isINFOCODE();
	  public void setSUCCESS();
	  public void setFAILED();
	  public void setINFOCODE();
	  public String getPrefix();
	  public String getBody();
	  public String getCode();
	  public void setPrefix(String prefix_);
	  public void setBody(String body_);
	  public void setCode(String code_);
	  public String getString();
	  public String to_String();
	  public boolean from_String(String returncode_str_);
	}

6.3 Essentials of the Distribution demo

The distribution demo consists of the management layer which is capable of distributing an application, and a demo application which is "Distributable", namely the HelloWorldApplication. It has a proxyable client, HelloWorldClient. They are listed in Appendix A. The management layer for the demo includes the ManagementServer that will be presented in Subsection 6.3.1 and Appendix A, the DistributionDimensionServer and the ManagementPolicy which will be presented in Subsection 6.3.3 and Appendix A. The system also needs the management client software that is able to work on the client side, namely the ManagementClient. It will be presented in Subsection 6.3.2. The Binding Object which implements the Network Transparency in the TCP/IP network is needed between the objects on server side and client side. It will be presented in Appendix A only. With the class library presented in Section 6.1, and the example implementation of the utility classes in Appendix A, the whole software will be presented.

6.3.1 Management Server

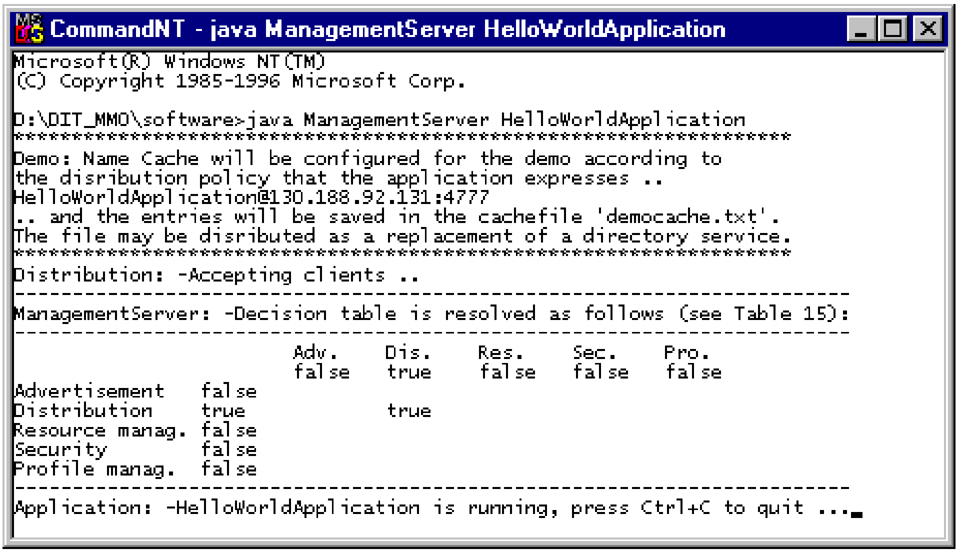
The partial source code listing below presents the ManagementServer class. The ManagementServer class implements interface "Management". The code reflects, in a precise manner, the class model in Figure 16 and the dynamic model in Figure 17 with its sub-state machines. The essentials of the management features of the system are presented. The code will be commented on in Appendix A.

```
import fi.vtt.ele.management_framework.implementation.management_layer.*;
import fi.vtt.ele.management_framework.implementation.application_layer.*;
import fi.vtt.ele.management_framework.implementation.utility.*;
import java.io.*;
class ManagementServer implements Management, Runnable
{
  public withPolicy application_ = null;
  public Decision decision_table[][] = new Decision[5][5];
  public Dimension _to_support[] = new Dimension[5];
  public Policy _on_demand[] = new Policy[5];
  public ReturnCode _on_init = null;
  public ReturnCode _on_decoding = null;
  public ReturnCode _on_policy = null;
  public ReturnCode _on_idle = null;
  public int _dimension = 0;
```

```
private String _UIN_ = null;
private String signature = null;
public Policy management_policy = null;
private int state = INITIALISING;
private int sub_state = 0;
/**@param _from_UIN_ is the application name to be started */
public ManagementServer(String _from_UIN_) throws Exception
{
  _UIN_ = _from_UIN_;
  state=INITIALISING;
  _on_init = createApplication(_from_UIN_);
  if (_on_init.isFAILED()) {exit(_on_init.getString());}
    else if (_on_init.isINFOCODE()) {exit(_on_init.getString());}
      else if (_on_init.isSUCCESS())
      {
        for (int i=0;i<=4;i++) _on_demand[i]=null;
        state=DECODING_POLICY;_
        on_decoding = decodePolicies(application_);
        if (_on_decoding.isSUCCESS())
        {
          for (int i=0;i<=4;i++) _to_support[i]=null;
          _dimension=0;
          state = CREATING_DIMENSIONS;
          for (_dimension=0; _dimension <= 4; _dimension++)
          {
            if (_on_demand[_dimension]!=null)
            {
              _to_support[_dimension] =
                         createDimensionServer(_on_demand[_dimension]);
            }
          }
          for (int i=0;i<=4;i++) for (int j=0;j<=4;j++)
                                            decision_table[i][j]=null;
          state=POLICYMAKING;
          _on_policy = policyMaking();
          printDecisiontable();
          if (_on_policy.isSUCCESS())
          {
            application_.run();
            state=IDLE;
          } else if (_on_policy.isFAILED())
                                          exit(_on_policy.getString());
              else exit(_on_policy.getString());
        } else if (_on_decoding.isFAILED())
                                      {exit(_on_decoding.getString());}
            else exit(_on_decoding.getString());
      }
}

public ReturnCode policyMaking()
{
  ReturnCode ret = getReturnCode();
  try
  {
    for (int i=0;i<=4;i++)
```

```
      {
      for (int j=0;j<=4;j++)
      {
         if ((_on_demand[i]==null) && (_to_support[j] == null))
          decision_table[i][j] = new DecisionImplementation
                                                 (false, false, false);
        if ((_on_demand[i]!=null) && (_to_support[j] == null))
         decision_table[i][j] = new DecisionImplementation
                                                  (true, false, false);
        if ((_on_demand[i]==null) && (_to_support[j] != null))
         decision_table[i][j] = new DecisionImplementation
                                                  (false, true, false);
        if ((_on_demand[i]!=null) && (_to_support[j] != null))
         decision_table[i][j] = new DecisionImplementation
                                                    (true, true, true);
      }
      }
      ret.setSUCCESS();
    } catch (Exception e) {ret.setFAILED();
                 ret.setBody("ManagementServer: -Policymaking failed.");}
    return ret;
  }

  public ReturnCode createApplication(String _from_UIN_)
  {
    ReturnCode ret = getReturnCode();
    Class check_ = null;
    try
    {
      File f = new File(_from_UIN_);
    } catch (NullPointerException e)
    { /** An object is not located */
      ret.setFAILED();
      ret.setBody(e.toString());
      return ret;
    }
    try
    {
      check_ = Class.forName(_from_UIN_);
    } catch (ClassNotFoundException f)
    { /** An object is not identified */
      ret.setFAILED();
      ret.setBody(f.toString());
      return (ReturnCode) ret;
    } /** An object is located and identified, but is it a one withPolicy?
*/
    try
    { /** Getting signature but not handling it in this version */
      application_ = (withPolicy)check_.newInstance();
      signature = application_.POLICY_SIGNATURE;
      ret.setSUCCESS();
    } catch (Exception g)
    {  /** An object is not withPolicy */
      ret.setINFOCODE();
      ret.setBody(g.toString());
    }
    return (ReturnCode) ret;
  }
/*************** the snipped code ends here *****************/
```

Figure 19 presents the screenshot captured during the test run of the system. The Management Server is started by expressing the name of the application. Then the distribution service is started by the command of the Management Server after the Management Server has resolved the policy. The distribution service has configured the local name cache for the demo simultaneously informing of the configuration - this particular service can be located at this particular address. It has reached the necessary information about the name of the service and the door at where the service will be available, from the application's distribution policy. The Management Server has printed the decision table. The table shows that the application is "Distributable" and the distribution service have been started successfully. The management layer objects will output more information about their states when the first client comes in.



```
Microsoft(R) Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.

D:\DIT_MMO\software>java ManagementServer HelloWorldApplication
*******************************************************************
Demo: Name Cache will be configured for the demo according to
the disribution policy that the application expresses ..
HelloWorldApplication@130.188.92.131:4777
.. and the entries will be saved in the cachefile 'democache.txt'.
The file may be disributed as a replacement of a directory service.
*******************************************************************
Distribution: -Accepting clients ..
------------------------------------------------------------------------
ManagementServer: -Decision table is resolved as follows (see Table 15):
------------------------------------------------------------------------
                        Adv.    Dis.    Res.    Sec.    Pro.
                        false   true    false   false   false
Advertisement   false
Distribution    true            true
Resource manag. false
Security        false
Profile manag.  false
------------------------------------------------------------------------
Application: -HelloWorldApplication is running, press Ctrl+C to quit ...
```

Figure 19. A screenshot of executing the Management Server.

6.3.2 Management Client

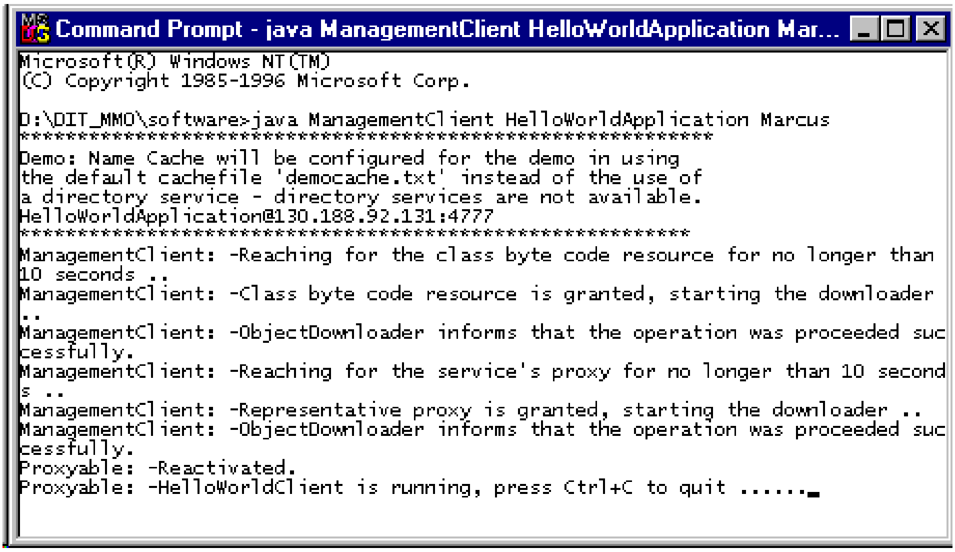
The partial source code listing below presents the ManagementClient class. The ManagementClient class implements the interfaces "Dimension" and "Distribution". The code reflects the class model in Figure 16. The code will be commented on in Appendix A.

```
import fi.vtt.ele.management_framework.implementation.management_layer.*;
import fi.vtt.ele.management_framework.implementation.application_layer.*;
import fi.vtt.ele.management_framework.implementation.utility.*;
import fi.vtt.ele.management_framework.implementation
.distribution_transparency.*;
public class ManagementClient implements Distribution, Dimension
{
  public Proxyable reconstructed = null;
  private int state = INITIALISING;
  private int REACHING_RESOURCE = 1000;
  private int dimensions = ROOT+DISTRIBUTION;
  private Identity client = null;
  private Identity service = null;
  private NetworkTransparency bo = null;
  private Messager messager = null;
  public ManagementClient(Identity service_, Identity client_)
  { service = service_;
    client = client_;
    try
    { bo = new TCPIPNetworkTransparencyBO();
System.out.println
("************************************************************");
System.out.println("Demo: Name Cache will be configured for the demo in
using   ");
System.out.println("the default cachefile 'democache.txt' instead of the
use of");
System.out.println("a directory service - directory services are not
available. ");
      bo.loadNameCache("democache.txt");
      bo.listNameCache();
      bo.setAddress(client_, new AddressImplementation(null,0));
System.out.println
("************************************************************");
      if (initMessager(client_, bo))
      {
        state = REACHING_RESOURCE;
        if (reachService(service_).isSUCCESS())
        {
          while (state != REACHING_PROXYABLE) {Thread.yield();}
        }
        if (reachService(service_).isSUCCESS())
        {
          while (state != RUNNING_PROXYABLE) {Thread.yield();}
          reconstructed.run();
        }
      } else exit("ManagementClient: -The service '"+service.to_String()
                                             +"' could not be reached.");
    } catch (Exception e)
                   {exit("ManagementClient: -Could not be installed.");}
  }
  public boolean initMessager(Identity _for_, NetworkTransparency bo_)
  { try
    {
      messager = new Messager(_for_, bo_, null);
      return true;
    } catch (Exception e) { return false; }
  }
  public boolean dispatchMessage(Message message_) { return false; }
  /** @param _action_ is a ManagementAction to handle. */
  public boolean handleAction(ManagementAction _action_)
  {
    if (state == DOWNLOADING_OBJECT)
    {
      if (_action_.isSUCCESS()) System.out.println("ManagementClient: "
      +"-ObjectDownloader informs that the operation was proceeded
                                                         successfully.");
      if ((_action_.isSUCCESS())&&(_action_.isREPRESENTATIVE_PROXY()))
      {
        Proxyable p = (Proxyable)_action_.getObject();
        reconstructed = p.reConstruct();
        if (reconstructed == null) exit("ManagementClient: "
                      +"-Representavive proxy cannot be reconstructed.");
        state = REACTIVATING_PROXYABLE;
        if (reconstructed.reActivate())
        {
          state = RUNNING_PROXYABLE;
        } else exit("ManagementClient: "
                        +"-Representavive proxy cannot be reactivated.");
      } else
      if ((_action_.isSUCCESS())&&(_action_.isCLASS_BYTE_CODE_RESOURCE()))
      {
        DiskFile p = (DiskFile)_action_.getObject();
        if (!p.toDisk()) exit("ManagementClient: "
                      +"-Class byte code resource cannot be installed.");
        state = REACHING_PROXYABLE;
      } else exit("ManagementClient: -Object cannot be downloaded.");
      return true;
    } else { return false;}
  }
  public ReturnCode reachService(Identity _of_)
  {
    ReturnCode sendquery = null;
    ManagementAction action = null;
    Message coming = null;
    try
    {
      sendquery = new ReturnCodeImplementation();
    } catch (Exception e) {exit("ManagementClient: -Could not create
object.");}
```

```
 if (state == REACHING_PROXYABLE)
 {
   sendquery.setQUERY();
   sendquery.setREPRESENTATIVE_PROXY();
   sendquery.setINFOCODE();
   if (sendMessage(_of_, sendquery))
   {
     messager.setTIMEOUT(10000);
     System.out.println("ManagementClient: "
+"-Reaching for the service's proxy for no longer than 10 seconds ..");
     coming = messager.receiveMessage();
     messager.setTIMEOUT(0);
     if (coming.isGRANTED())
     {
       System.out.println("ManagementClient: "
         +"-Representative proxy is granted, starting downloader ..");
       try
       {
         action = new ManagementActionImplementation();
         action.adaptMessage(coming);
         Thread t = new Thread(new ObjectDownloader(action, bo, this));
         state = DOWNLOADING_OBJECT;
         t.start();
         action.setSUCCESS();
         return action.createReturnCode();
       } catch (Exception e) {return action.createReturnCode();}
     }
   } else if (coming.isDENIED()) { exit("ManagementClient: "
        +"-Representative proxy is denied by the server, quitting.");}
   else { exit("ManagementClient: -Cannot send a message.");}
 }
 if (state == REACHING_RESOURCE)
 {
   sendquery.setQUERY();
   sendquery.setCLASS_BYTE_CODE_RESOURCE();
   sendquery.setINFOCODE();
   if (sendMessage(_of_, sendquery))
   {
     messager.setTIMEOUT(10000);
     System.out.println("ManagementClient: -Reaching for the class
                byte code resource for no longer than 10 seconds ..");
     coming = messager.receiveMessage();
     messager.setTIMEOUT(0);
     if (coming.isGRANTED())
     {
       System.out.println("ManagementClient: "
      +"-Class byte code resource is granted, starting downloader ..");
       try
       {
         action = new ManagementActionImplementation();
         action.adaptMessage(coming);
         Thread t = new Thread(new ObjectDownloader(action, bo, this));
         state = DOWNLOADING_OBJECT;
         t.start();
         action.setSUCCESS();
```

```
            return action.createReturnCode();
          } catch (Exception e) {return action.createReturnCode();}
        }
      } else if (coming.isDENIED()) { exit("ManagementClient: "
                        +"-Class byte code resource denied, quitting.");}
      else { exit("ManagementClient: -Cannot send message.");}
    }
    return action.createReturnCode();
  }
/*************** the snipped code ends here ****************/
```

Figure 20 presents the screenshot captured during the test run of the system. Management Client was started and it had connected to the given application (served by a management server object). The class code resource of the representative proxy was required first and it was granted, then the representative proxy object of the example application was required, and it was granted, too. As soon as all the necessary data was reached, the representative proxy object was started successfully.


Command Prompt - java ManagementClient HelloWorldApplication Mar...

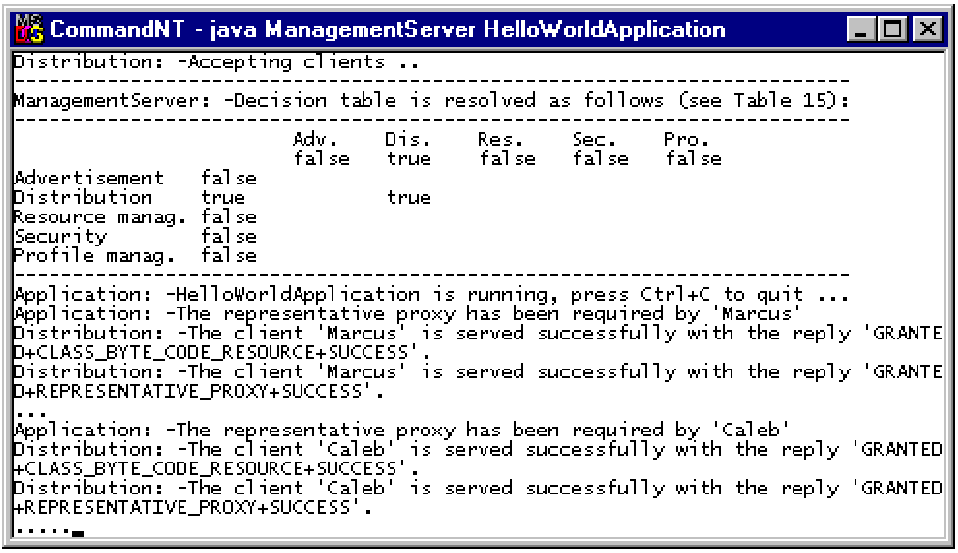
```
Microsoft(R) Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.

D:\DIT_MMO\software>java ManagementClient HelloWorldApplication Marcus
******************************************************
Demo: Name Cache will be configured for the demo in using
the default cachefile 'democache.txt' instead of the use of
a directory service - directory services are not available.
HelloWorldApplication@130.188.92.131:4777
****************************************************
ManagementClient: -Reaching for the class byte code resource for no longer than
10 seconds ..
ManagementClient: -Class byte code resource is granted, starting the downloader
..
ManagementClient: -ObjectDownloader informs that the operation was proceeded suc
cessfully.
ManagementClient: -Reaching for the service's proxy for no longer than 10 second
s ..
ManagementClient: -Representative proxy is granted, starting the downloader ..
ManagementClient: -ObjectDownloader informs that the operation was proceeded suc
cessfully.
Proxyable: -Reactivated.
Proxyable: -HelloWorldClient is running, press Ctrl+C to quit ......
```

Figure 20. A screenshot of executing the ManagementClient.

Figure 21 shows what happens on the server side during the interactions of clients. It shows that the application object has been notified about the client's identity. The application object cannot control the action directly, but through the policy that the ManagementServer maintains. However, the application object can distinguish individual clients.

It must be notified that the reaching byte code resource for the proxy object only, is not usually sufficient - the proxy object may consist of many other application specific classes and even may use another class library than what is currently installed in the target platform - and the resource management dimension services must therefore take place. This subject will be studied further in Chapter 7.

All accessing in the system is made only by giving identities of entities. In the future version, the expression of the identity will be expanded also to carry the less-friendly expression of UID like 'BE9CD630-A96B-11D3-81BC-00902790ECE5' to distinguish identities properly.



```
Distribution: -Accepting clients ..
------------------------------------------------------------------------
ManagementServer: -Decision table is resolved as follows (see Table 15):
------------------------------------------------------------------------
                         Adv.    Dis.    Res.     Sec.    Pro.
                         false   true    false    false   false
Advertisement    false
Distribution     true            true
Resource manag.  false
Security         false
Profile manag.   false
------------------------------------------------------------------------
Application: -HelloWorldApplication is running, press Ctrl+C to quit ...
Application: -The representative proxy has been required by 'Marcus'
Distribution: -The client 'Marcus' is served successfully with the reply 'GRANTE
D+CLASS_BYTE_CODE_RESOURCE+SUCCESS'.
Distribution: -The client 'Marcus' is served successfully with the reply 'GRANTE
D+REPRESENTATIVE_PROXY+SUCCESS'.
...
Application: -The representative proxy has been required by 'Caleb'
Distribution: -The client 'Caleb' is served successfully with the reply 'GRANTED
+CLASS_BYTE_CODE_RESOURCE+SUCCESS'.
Distribution: -The client 'Caleb' is served successfully with the reply 'GRANTED
+REPRESENTATIVE_PROXY+SUCCESS'.
.....
```

Figure 21. A screenshot showing the ManagementServer command window after two clients have been served.

Before use, the name cache must be configured accordingly, to express physical addresses for the service identities - usually this kind of name service is provided by a directory service in the distributed system. In the demo, the Management Server configures the file "democache.txt" by which the target platform then should be provided. For testing the system in one machine, it is not necessary to pass the cache file anywhere. The file may be configured to contain information for many services in different locations. The binding object that implements the network transparency must take care of accessing the name service to maintain its cache. However, the name cache is needed to make the resolving of the names faster.

Usage of the client demo: 'java ManagementClient <service_UIN> <client_UIN>, where the <service_UIN> is the service's unique name, e.g. HelloWorldApplication and the <client_UIN> is the user's unique name, e.g. Marcus. The application <service_UIN> must have been started with management before using this client demo.

6.3.3 Management Policy

The snipped source code below presents the structure of the ManagementPolicy class used in the demo. All policy definitions have been left open for further development - this policy can maintain the policy that the example application carries - and a more generic way to handle the policies must be developed in the next version.

```
import fi.vtt.ele.management_framework.implementation.utility.*;
import fi.vtt.ele.management_framework.implementation.management_layer.*;
import fi.vtt.ele.management_framework.implementation.utility.*;
import fi.vtt.ele.management_framework.implementation.management_layer.*;
/** This class is created only for the test purpose. This is the place in
where the descriptions of the policy must be developed as a further
study.*/
public class ManagementPolicy implements Policy
{ public int DIMENSION = Dimension.ROOT;
  /** The next definitions are ROOT-related. */
  public String _UIN = null;
  /** UID can be produced by the common GUIDGEN-utility. */
  public String _UID = null;
  /** The next definitions are DISTRIBUTION-related. */
  public int PREFERRED_DOOR = 0;
  public int MAX_NUMBER_OF_REPRESENTATIVE_OUT = 0;
  public ManagementPolicy() throws Exception { }
  public ManagementPolicy(InformationBranch from_) throws Exception
  {  DIMENSION = DIMENSION+from_.getDIMENSION();
    _UID = from_.getUID();
    _UIN = from_.getUIN();
    PREFERRED_DOOR = from_.getPREFERRED_DOOR();
    MAX_NUMBER_OF_REPRESENTATIVE_OUT =
getMAX_NUMBER_OF_REPRESENTATIVE_OUT();
  }
  public int getDIMENSION() { return DIMENSION; }
  public String getUIN() { return _UIN; }
  public String getUID() { return _UID; }
  public int getPREFERRED_DOOR() { return PREFERRED_DOOR; }
  public int getMAX_NUMBER_OF_REPRESENTATIVE_OUT()
  {
    return MAX_NUMBER_OF_REPRESENTATIVE_OUT;
  }
}
```

6.4 Summary

In this chapter, the implementation of the solution was presented in using the Java programming language. The solution is not an endproduct, however, therefore to achieve a presentation that is clear enough, the essentials of an example application were presented. Programmers, who may implement the applications of this framework in the future, can use the example as a template. The goal was to make the solution explicitly clear. The applicable class library was also presented, because the solution is meant to be deployed in deploying the class library. There was a possibility to produce the results of the implementation by using proper UML-tool software by deriving the design models and then modifying them to use Java and eventually generate the necessary piece of codes with the code generation tools shipped with the tool. However, this was decided to be left as an initiative for further development. Deployment of the presented solution can be done by producing applications separately from the management layer. There can be a different management layer implementation for different platforms. Any compatible application can then deploy any of these management supports in terms of the management policy of the application.

7. Discussion and further research

The following issues will be considered: 1) What is the answer to the research problem? 2) Which advantages can be considered as achieved due to the solution? 3) Which subjective quality properties the system can be observed to support? 4) Which are weaknesses of the solution? 5) Which are further development issues of the system? 6) Can an example of further design of the system be provided?

7.1 Answer to the research problem

What is the answer to the research problem? Originally, the objectives were concentrated on in Chapter 1, Subsection 1.5 by setting the research problem. The answer is provided in Table 20.

Table 20. Answer to the research problem.

Problem		Answer
A framework to manage the distributed use of a piece of computer software has to be developed. *(continues)*	The user requirements must be examined.	The preferred user requirements are listed in Table 2 and the threshold user requirements are listed in Table 13.
	Suggestions presented in Table 1 must be evaluated, and if the dimensions can be proved to exist, and as a consequence - further management functions to be isolated, then those functions must be developed further, this thereby leading to the implementation.	Suggested ideas have been found most relevant in considering the problem. The dimensions have been proved to exist in Figure 4 by the correlation analysis. These factors were combined to define the concept. The concept is presented in Chapter 3.
The table continues.		

Continuing Table 20.

Problem		Answer
A framework to manage the distributed use of a piece of computer software has to be developed *(continuing)*.	The framework must be defined on the level of abstractions and further design models so it can be realised for many purposes, for instance as a test platform that is easy to understand and deploy.	The architecture diagrams in Figure 12 and Figure 13 represent the abstractions of the system. Organising the system by layers and dimensions presented in Table 15 supports the *understandability* of the system.
	APIs of the system must be defined.	Abstractions of the APIs are presented in Table 15, and the essential features of them are realised in Figure 15 and Figure 16.
	A demo program for deploying the solution must be presented.	The *ready-to-run* demo has been presented in the Appendix A.

7.2 Advantages of the solution

Which advantages can be considered as achieved due to the solution? The highlights of the developed technical solution are:

- Computational objects and their possible representatives can be fully managed in their environments.

- A computational object can be accessed only in using its unique identifier - we do not have to care about where the objects actually are located - the location transparency (as a sub set of the distribution transparency) takes care of that responsibility.

- There can be different binding objects implementing the network transparency for the different circumstances, and they can be made manageable, too. The same computational objects will work everywhere because all they have to know is the interface of the binding object - the network transparency - and the interface will always stay the same. As a further study, the author of this thesis would like to suggest the binding object to be provided in using DDE, WAP, MExE, or Bluetooth when their corresponding network transport layer and the name service capabilities are deployed.

- A simulating development environment can be provided by realising the interfaces only for emulating objects of the real world.

- Manageable binding objects can be instantiated for the application layer as well - the application and its representative client can communicate in the same way by using only the identities of the objects.

The introduced management framework can be deployed with any data objects, whether these are passive or active (active data objects include program logic). Every particular data object can be made manageable if they can express the policy of how they are expecting to be managed. The policy definitions must be made flexible, so the applications can be adapted for every individual case. This issue will be further dealt with in Section 7.5.

The solution opens enormous possibilities to manage software objects in their environment, whether they are distributed or not. It can be stated that all the preferred user requirements introduced in Table 2 can be fulfilled. To justify this statement, an example will clarify the matter: There was a requirement with the label PE11. Indeed, with the solution, it is possible to provide a vision in which the copying of data objects is explicitly managed. Making of copies can be authorised, secured limited etc. These definitions may have been written in the policy information of the application.

7.3 Subjective quality properties of the system

Which subjective quality properties the system can be observed to support? Many of these properties were examined by means of the practical systems study in Chapter 2. The solution is a framework and because the subjective quality properties can exist only with a real product, many of these properties will be seen only in the future with applications based on the solution.

There is a large category of different objectives and categorised requirements connected with the RM-ODP standards Part1 [17], and there are the Threshold requirements in Table 13. Many of these requirements cannot be seen yet, many of them will be realised when there is realised solutions of the different dimensions of the distributed system. However, to provide an estimation, a more compact category from Colouris et al. [16] will be used together with another, more practical category seen as the Jini design centre discussed in Table 12. These will be evaluated against the solution, and the results will be presented in Table 21. The source of every individual characteristic is mentioned.

Table 21. Estimation of the common key characteristics of the distributed system existing in the solution.

Characteristic	Estimation notes
Resource sharing (Colouris et al.)	Maximal resource sharing will be reached. Every piece of resource can be shared as a service for others. The device that offers its service is represented by the proxy object, which makes the deployment of the service easy.
Openness (Colouris et al.)	The solution is open, because it is available in public. The term of "openness" also considers the uniform inter-process communication mechanism to access the shared resources, and that the system can be constructed from heterogeneous hardware and software, possibly from different vendors. In the solution, these can be supported by implementing the corresponding functions of the dimension interfaces.
The table continues.	

Continuing Table 21.

Characteristic	Estimation notes
Concurrency (Colouris et al.)	Concurrent sessions of management clients are supported. An application supported with its own set of management objects can run concurrently even on the same platform. The application object itself has to support its multiple representatives and their concurrent actions.
Scalability (Colouris et al. and the Jini design centre)	The system is scaleable. One dimension can also be divided to be processed in more than one process. The selection will be driven by the policy of the application. The advertisement and the distribution dimensions of the solution together fulfil the same scalability feature as Jini's lookup service does by grouping the services into communities and the communities further into federations.
Fault tolerance (Colouris et al.)	With the implementation of the resource management dimensions services and the failure transparency features, the system will tolerate all kinds of faults.
Transparency (Colouris et al.)	The local mode and the distributed mode of the solution differ from each other only by distribution transp. support.
Simplicity (the Jini design centre)	Minimal software is needed to be added on e.g. Java, as is the case with Jini.
Reliability (the Jini design centre)	The distribution transparency and the resource management dimension services of the solution will add value on Jini's reliability scheme.
Device agnosticism (the Jini design centre)	With Jini's proxy-like representatives, the solution fulfils the same device agnosticism scheme as Jini does.

7.4 Weaknesses of the solution

Which are weaknesses of the solution? The full deployment of the solution acquires an access to every data object for them to be checked by the management layer. The state diagram of the management server class in Figure 18 represents the necessary checking protocol. To guarantee management, the protocol checks the management requirements of an object, and allows or disallows the execution according to the policy of an object.

It seems almost utopistic to expect that the checking protocol can be supported among the providers of the operating systems on a necessary scale. An official standardisation or a very large consortium is needed to support the solution. This issue will be covered in Section 7.5.

7.5 Further research issues

Which are further development issues of the system? It has been said in the research problem analysis that "the solution can be used when R&D of distributed software continues further concentrating to examine the fields of problems that will exist with the upcoming wireless and mobile communication technology". Next, the initiatives of this R&D will be focussed on. The initiatives are listed below:

- Develop the management policies that an application must carry - expressing them is the essential factor of the system. They must be defined by a large consortium to achieve full advantages of the solution.

- Develop the technique concerning how the management policies could be formed, partially by the automatic operation between the systems, partially by the manual operation between stakeholders, both according to their associated management domains.

- Develop the generic and implementation independent management and algorithms (some kind of protocols). They will be the bases from which the management feature in new generations of implementation would be derived.

- Develop the Advertisement dimension service to follow the advertisement policy that the application object explicitly expresses.

- Develop the Security dimension service to follow the security policy that the application object explicitly expresses.

- Develop the implementation of the distribution transparencies interfaces further, e.g. for using the DDE-derived technique, or possibly, for using the communication technique used in the wireless and mobile environments, e.g. MExE, WAP, or Bluetooth (if the platform can communicate, the distribution transparency could be mounted on it).

- Develop the Resource management dimension service to follow the resource management policy that the application object explicitly expresses. This will include at least developing:

    - the manageable resource instance interface of "Manageable", and its functions including:

        - how to get the resource reservation tree,

        - how to get the dependencies existing between the resource instances - the dependency tree will be formed,

        - how to get the resource management policy automatically, and

        - how to apply the management action to the resource instance,

    - the technique to adjust the load (load balancing) between the server and the client,

    - the techniques to cache the resource instances, and

    - a resource management negotiation technique.

- Develop the profile management dimension service. The profile management dimension does not exist in the models, but it is supported in the class library. This new dimension should be added in the system in the future versions.

- Apply the framework to the existing distributed objects and components.

- Apply the framework to the existing executable program modes.

- Transfer the models into an advanced UML-tool.

- Start the SCM-activities [64], e.g. the version control. The configuration items in this thesis belong to the version 1.0.

- Start the PLA-associated activities (Product Line Architecture) to support the product differentiation taking place in the future [65].

7.6 An example of further design - the Resource management

Can an example of further design of the system be provided? During the process, the development of the resource management dimension was also started. This subject had to be left out of the design because of the time schedule and too large a document. It was decided that the subject would be one of the further research initiatives. Nevertheless, the results of this unfinished work will be included here because they are considered significant.

The application mode which is "Resourceable" and the management layer, which is capable to maintain it, will be presented. Only the static models will be presented. They may have been modified heavily when the actual design of the resource management takes place.

7.6.1 Resources of the distributed systems

An application uses the resources of its execution platform. The resources are successfully allocated once, after the application has been started. The resources outside the local execution platform do not exist because they are served as services, being already managed on their original installation points. Only the piece of software that replaces the abstract of "Application" in Figure 22, must be managed here (with its possible representatives, of course). A situation in which the resource instance might be lost totally or might go below the QoS criteria set by the policy, must be invisible to the end-user as long as possible. This is also one of the purposes of the distribution transparencies in the distributed environment. There may exist a situation where a defined condition can not be maintained, an example includes losing the network connection for a long time (the time will be set by the policy) - then the malfunction is fatal, and the end-user and associated representatives of the other roles involved must be informed about the error.

7.6.2 Class diagram of a "Resourceable" application

Figure 22 illustrates the static design model of the application that can carry the information about how it wants to be managed.

All interfaces presented earlier are still valid. The application can express its requirements of the resource management by implementing the function of "getResourceManagementPolicy ()" of the interface of "Resourceable". The resources that the application has allocated successfully exist as "Resource Instance". Every individual instance may or may not be "Manageable". The manageable properties have usually been derived from the resource source, as is the case with the OO-class libraries. The programmer of the application has to make the choice whether to use manageable resources or not. When using manageable resources, the programmer also defines the static management policy at the programming time. The definitions consider the QoS requirements as well. The individual resource instance can be dependent on the other resource instances, which may or may not be manageable. A bundle of resource instances can exist, and these can belong to the one policy that covers the whole bundle.

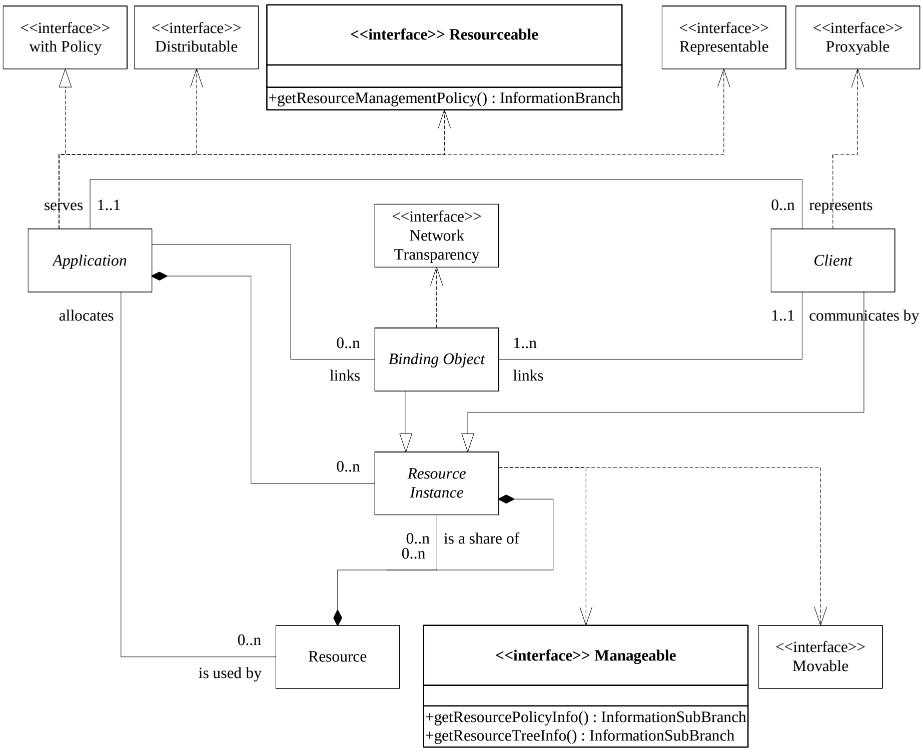
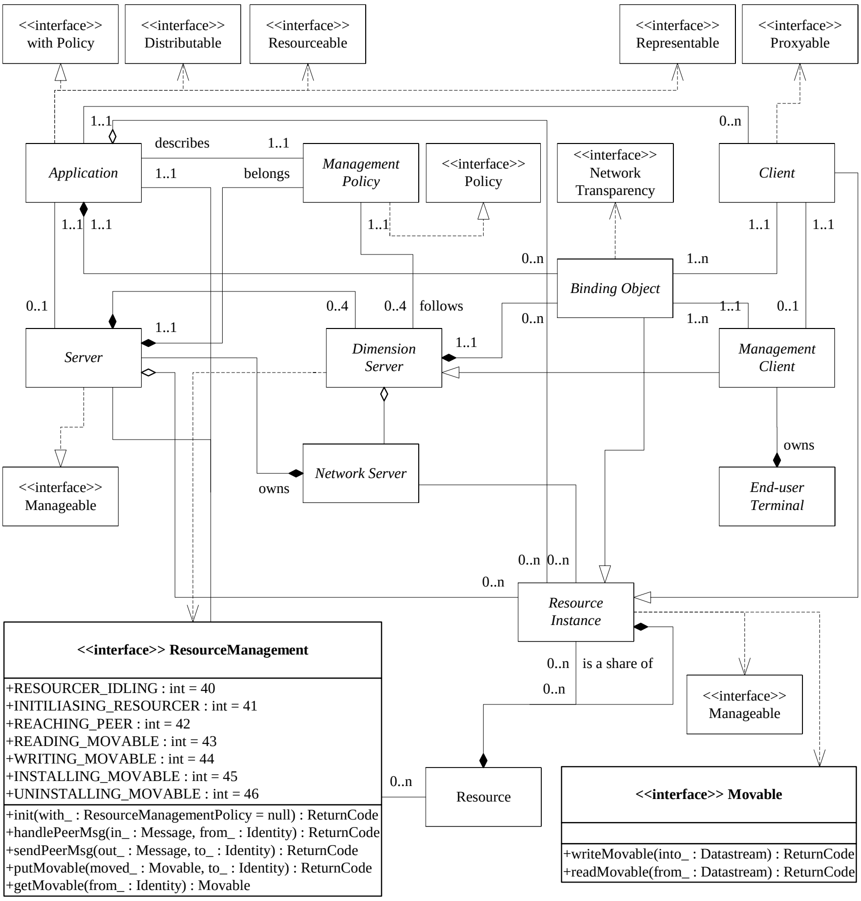

<<interface>> with Policy
<<interface>> Distributable
<<interface>> Resourceable
+getResourceManagementPolicy() : InformationBranch
<<interface>> Representable
<<interface>> Proxyable
serves
1..1
0..n
represents
<<interface>> Network Transparency
Application
Client
allocates
1..1
communicates by
0..n
Binding Object
1..n
links
links
0..n
Resource Instance
0..n
is a share of
0..n
0..n
Resource
is used by
<<interface>> Manageable
+getResourcePolicyInfo() : InformationSubBranch
+getResourceTreeInfo() : InformationSubBranch
<<interface>> Movable

Figure 22. Class model for a "Resourceable" application.

7.6.3 Class diagram of the Resource management dimension layer

Figure 23 illustrates the static design model of the management layer. All interfaces presented earlier are still valid.

The resource management will be handled by the "Resource Management Dimension Server" object assisted with the "Resource Management Client" object. A need to move the resource instance from one place to another may exist. Therefore, the "Resource Instance" object must be able to be made "Movable". Only the software based resources can be moved, the physical (hardware based) resources can not. However, the hardware-based resources are usually used by a control interface, and the control interface can be made into a remote control service and thereby can be available for other applications.



<<interface>> with Policy
<<interface>> Distributable
<<interface>> Resourceable
<<interface>> Representable
<<interface>> Proxyable
Application
describes
belongs
Management Policy
<<interface>> Policy
<<interface>> Network Transparency
Client
Binding Object
follows
Server
Dimension Server
Management Client
Network Server
owns
owns
<<interface>> Manageable
End-user Terminal
Resource Instance
is a share of
<<interface>> Manageable
<<interface>> ResourceManagement
+RESOURCER_IDLING : int = 40
+INITILIASING_RESOURCER : int = 41
+REACHING_PEER : int = 42
+READING_MOVABLE : int = 43
+WRITING_MOVABLE : int = 44
+INSTALLING_MOVABLE : int = 45
+UNINSTALLING_MOVABLE : int = 46
+init(with_ : ResourceManagementPolicy = null) : ReturnCode
+handlePeerMsg(in_ : Message, from_ : Identity) : ReturnCode
+sendPeerMsg(out_ : Message, to_ : Identity) : ReturnCode
+putMovable(moved_ : Movable, to_ : Identity) : ReturnCode
+getMovable(from_ : Identity) : Movable
Resource
<<interface>> Movable
+writeMovable(into_ : Datastream) : ReturnCode
+readMovable(from_ : Datastream) : ReturnCode
1..1
1..1
1..1
1..1
1..1
1..1
1..1
1..1
1..1
1..1
1..1
1..1
0..n
0..n
0..n
0..n
0..n
0..n
0..n
0..n
0..n
1..n
1..n
0..1
0..1
0..4
0..4

Figure 23. Class diagram of the management layer with the Resource management support.

7.7 Summary

In this chapter the solution has been analysed carefully from different viewpoints concerning the answer to the research problem, the advantages of the solution, the subjective quality properties of the system, the weaknesses of the solution, the further research issues, and the example of the further design.

The design of the system was started with setting the criteria for the objective quality by examining the user requirements. Thus the goal was known, and thereby it was possible to define the answer to the question "what should we do?" The requirements together with the answer were used to drive the selection of the choice concerning the different technical properties of the system during the whole design process. Whether the original requirements will be realised in the application based on the management framework solution here is an issue that will be seen later on, but this may well happen.

At the beginning of the document, the subjective quality properties of a software product were discussed. They were examined them and consequently, the system was designed bearing the subjective quality factors in mind. Thus, the question "how will we do it?" was answered. Now it needs to be discussed whether there is any subjective quality present in the solution.

Because the solution takes the form of a framework, the usefulness of the solution can be evaluated only by evaluating the applications the management of which is based on the realisation of the framework. Still, it was possible to evaluate whether the solution will be able to realise the objectives and have a set of common design criteria for the distributed system achieved. It could be seen that the objectives and the common criteria would be fulfilled, as this evaluation is possible at this stage. It was found out that the weakness of the solution lies in the need to standardise the management checking protocol. The state chart of the Management Server object presents the necessary protocol in Figure 18. However, even without any standardisation process, the solution will be useful in the context of its original objectives.

Furthermore, a significant set of further research initiatives was presented.

8. Conclusions

When software products become a more and more important part of our everyday life, the software products and information services should be treated just like usual consumables - according to the laws and rules set by society, and equally, according to contracts between end-users and content or service providers. However, the way to trade software products and services differs from these usual consumables - software business is mostly made in the global communication network - the Internet. The way to include attributes of the contracts between different stakeholders considering the software products must be defined, and equally, the technique to guarantee the management of the software application in the end-user's environment must be developed. In this thesis, software application is understood to be any piece of software that can be provided, from deploying one byte to deploying a mail service - large systems of that kind are usually provided by deploying some component model. Because the different component and object models are building parts of software applications, the technique to be developed must be able to support them in particular.

The technology to be used should have a self-configurability feature - products should maintain themselves when used in different circumstances. The end-user should not have to worry about whether the software used has to be updated, or even whether the software and hardware resources in his terminal equipment are updated and working - the technology must provide the maintenance without any end-user intervention. The end-user should always have, even unknowingly, updated software and even updated hardware support - this may also be a question of contracts. Furthermore, does the end-user have to be bound to some individual device to maximise the deployment of technology, as is the case with our desktop computer today? This does not need to be the case; instead, the end-user should be able to deploy the application with his personal preferred configuration wherever he happens to be located and whatever device he happens to use. Let us take an example: the end-user would like to use his banking services for checking the account balance - therefore he just takes some terminal equipment, a public kiosk terminal, a cell phone or whatever compatible device happens to be near, and will be able to reach and deploy the service wanted with personal settings for the device type of the terminal and with personal settings for the service used - this, moreover, with great security and usability. The analysis of the preferred user requirements the ways they can be seen are listed below:

End-user:

- End-user can easily reach and deploy a software product or information service by using any compatible terminal equipment according to a contract.
- End-user gets his personal profiles for the equipment.
- End-user gets his personal profiles for a service.
- End-user has data and access security maintained according to a contract.
- End-user will be provided a vision of the information environment that offers easy-to-use, and even profitable applications and services.

Society:

- Software products or information services are considered as usual consumables (e.g. in relation to marketing regulations, product liability or data security).

Device provider:

- Device provider can provide compatible terminal equipment with maintenance according to a contract.

Service provider:

- Service provider can provide rich applications types the way end-users are provided with today, but in a maintained mode.
- Service provider can provide software products or information services with maintenance according to a contract.
- Service provider can do business by trading software products and information services with contracts.

Content provider:

- Content provider can maintain the authorising and licence information with provided software products or information services according to a contract.

A software product must be able to include management policy - and the underlying technique must be able to follow the policy, or back down if the maintenance cannot be guaranteed in terms of the policy. The structure of the policy must be defined the way that it considers different management domains and the domains can intersect.

This thesis sets a framework for the management technique in the context depicted above.

The problem has been approached by setting the definitions as follows:

- "Objective quality thinking - the system quality must be ensured before actually doing anything".

- "If we will find out the requirements for the largest system, we will know which the requirements for the smallest system might be - then we will simply develop the management model for any piece of the software by deploying the requirements found".

Moreover, to apply creative thinking to the process, some suggestions, assumptions, and questions concerning the solution were made as follows:

Client-server model:

- Application program should be such that only the part of an application that is absolutely needed to present the application to the end-user will be loaded onto a memory of a terminal device. This loadable part of the application would work as a bridge between itself and its user, the application itself still being able to remain anywhere in the network. The application itself could be as large as necessary and be combined from some other services. End-user should see only the part of the application that is necessary for using it, and the user may pay only by usage. This suggestion follows the idea of a proxy-object currently used in the Jini and Microsoft.NET remoting concepts.

Resourcing:

- When loading it, it should be taken care of how to configure and arrange resources for the application to enable its working in the end-user terminal device.

- A multiple allocation of resources can be avoided when every individual service exists only once in some specific interior.

Decomposing:

- The application itself may in fact maintain its function group of security, advertisement, distribution and other maintenance according to a predefined strategy defined by the *policy*.

- Can those function groups in the previous item be pointed to be *orthogonal* compared to each another (concerning *dimensions* of software objects)?

- Can we prove the existence of the dimensions of software objects by exploring practical systems (introducing a new term to define a shorter description of the phrase 'practical technical feature' - a feature found in practical systems that is a candidate function to be included in a dimension - it is suggested to be called a *Mini-concept*)?

Layering:

- Applications programmers should program only the application logic, not the support for distribution or management - can we assume that there should be a *layer* of application level and the layer of management level included in underlying systems?

Distributed nature:

- Can the consequences of unstable distributed environment really be masked out totally to offer programmers to become isolated from the difficulties of programming for distributed environment? With this, we suggest to use the *distribution transparencies* introduced at least by the RM-ODP.

With the presented discussion in mind the system was began to develop. First, the concept of the technique to be realised was defined. To define the concept, the user requirements and the available technical features to be set to satisfy the requirements were analysed by using the QFD-matrix. The user-requirements were reached by deploying the Kano Model. To clarify the technical aspect - presented by the mini-concepts - the thesis provides a discussion on the techniques in the existing distributed system by which the user requirements have been demanded in the real world until now.

The best concept is defined as follows:

- do not decide upon the client-server model until the very last moment, maybe as late as in run time,

- divide the system into two layers - one to provide the support for applications, the other to provide support for maintenance,

- let the client called 'proxy' represent the application, so the client can be built to be as fat or as thin as necessary - and make it movable,

- support user and device profiles to maintain the preferred settings of the end-user,

- support information for management policy with many different levels of abstractions to maintain different viewpoints on distributed software,

- support the system of local- or distributed mode, apply the distribution transparencies to hide the distributed mode from applications,

- support dimensions and their corresponding policies,

- do not require management support as a mandatory feature, let application contain the information of whether full or partial management is needed in every particular case,

- support the old application mode (e.g. EXE-mode) by a voluntary extension of management information to be added in the case the old application modes would like to get management support, and

- consider the functions provided by a working application as a service for other entities of human or artificial origin.

The system architecture was developed based on the concept. The architecture was then realised into the design model of the system. The UML modelling technique was used. In utilising the UML, we reached a very important strategic advance if considering the future of the solution. In addition to the number of software products increasing, the demands toward the designers of software increase, too. A software application must be produced faster and faster, in a cheaper and cheaper manner, and still with more quality, to maintain, in the future, the product's position in the market from a point of view of the software business. By utilising the most advanced graphical tools, OO-methods and code generation technique it will be possible to achieve this goal. During the modelling phase, this thesis stayed on the abstract level of the solution. The implementation was carried out on the level of abstraction too - the presented APIs are abstractions. By this the ability of the management framework to be adopted in different environments and realised in using different programming techniques, is being supported. The class library to deploy the framework was produced and fully presented - by this, the openness of the solution was supported. When the solution is realised to take advantage of the management features, the APIs will be implemented in using proper programming tools, and supporting the different techniques in the underlying platforms and their communication capabilities. To assist the realisations, the example realisation is included in the thesis. The example realisation is as simple as possible still illustrating the purpose and use of the system. The realisation is intended to be associated to the research purpose.

In the chapter "Discussion and further research" this paper covered the issue of whether the solution is fulfilling the objectives. This was found to be in view. The introduced management framework can be deployed with any data objects, whether these are passive or active (active data objects include program logic). Every particular data object can be made manageable, if they can express the policy of how they are expecting to be managed. The highlights of the developed technical solution are:

- Computational objects and their possible representatives can be fully managed in their environments.

- A computational object can be accessed only in using its unique identifier - we do not have to care about where the objects are actually located - the location transparency (as a sub set of the distribution transparency) takes care of that responsibility.

- There can be different binding objects implementing the network transparency for the different circumstances, and they can be made manageable, too. The same computational objects will work everywhere because they have to know only the interface of the binding object - the network transparency - that will always stay the same.

- A simulating development environment can be provided by realising the interfaces only for emulating objects of the real world.

- The manageable binding object can be instantiated for the application layer as well - the application and its representative client can communicate in the same way by using only the identities of the objects.

The main purpose of the solution might refer to when R&D of distributed software continues further concentrating to examine the problematic issues that will exist with the upcoming wireless and mobile communication technology.

The significant set of the further research initiatives was presented. The most important initiatives are:

- to develop the management policies which an application must carry - expressing them is the key factor of the system,
- to develop the generic and implementation independent management and algorithms (some kind of protocols),
- to develop the resource management service to follow the resource management policy that the application object explicitly expresses, and
- to develop the implementation of the distribution transparencies interfaces further, e.g. for using the DDE-derived technique, or possibly, for using the communication technique used in the wireless and mobile environments, e.g. MExE, WAP, or Bluetooth.

To gain the management features support on a large scale, there is a heavy obstacle standing in the way; to guarantee management, the management protocol must check the management requirements of an object, and allow or disallow the execution according to the policy of an object. It seems almost utopistic to expect that the checking protocol can be supported among the providers of the operating systems on a necessary scale. An official standardisation or a very large consortium is needed to support the solution. After all, if extended to its full design and after the official or unofficial standardisation process, the solution would be a candidate for a whole new software management technology. However, the solution will be useful in the context of its original objectives.

I hope that at least some of the presented ideas and the research initiatives being applied to the distributed systems and software in the future.

References

1. Edwards, K. 1999. Core JINI. Sun Microsystems Press. 832 p.

2. Pressman, R. 2000. Software Engineering, a practitioner's approach. 5th Edition. Columbus, Ohio, USA. McGraw-Hill Companies. 840 p.

3. Day, R. 1993. Quality Function Deployment: Linking a Company with Its Customers. Milwaukee, Wisconsin. ASQC Quality Press. 245 p.

4. McCall, J., Richards, P. & Walters, G. Factors in Software Quality. In: US Rome Air Development Center Reports. NTIS AD/A-049 014, 015, 055. 1977. (RADC TR-77369 vols I, II, III.)

5. Iivari, J. 1991. A paradigmatic analysis of contemporary schools of IS development. European Journal of Information Systems 1, No 4. Pp. 249 - 272.

6. Järvinen, P. 1999. On Research Methods. Opinpaja Oy, Tampere, Finland. 129 p.

7. Akao, Y. 1990. Quality Function Deployment: Integrating Customer Requirements into Product Design. Productivity Press, Cambridge, MA, USA. 387 p.

8. Niemelä, E. 1999. A component framework of a distributed control systems family. Technical Research Centre of Finland, VTT Publications 402. 188 p. + app. 68 p.

9. Gamma, E., Helm, R., Johnson, R. & Vlissides, J.O. 1994. Design patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Publishing Company. 383 p.

10. Moilanen, M. 27.2.2001. The LONTONEXTG student project of University of Oulu and VTT Electronics. URL: http://www.iki.fi/~mmoi/studyprojects/tsproject/index.html.

11. Platt, D. 2001. Introducing Microsoft.NET. Microsoft Press. 240 p.

12. Blair, G. & Stefani, J.-B. 1997. Open Distributed Processing and Multimedia. Addison-Wesley Publishing Company. 452 p.

13. Swan, J.E. 1988. Consumer satisfaction related to the disconfirmation of expectations and product performance. Journal of Consumer Satisfaction, Dissatisfaction, and Complaining Behaviour, 1. Pp. 40 - 47.

14. Norton, P. 1998. Peter Norton's Essential Concepts, 3rd edition. McGraw-Hill Companies, Columbus, Ohio, USA. 242 p.

15. Norton, P. 1988. The New Peter Norton Programmers Guide to The IBM PC&PS/2, 2nd edition. Microsoft Press. 511 p.

16. Colouris, G., Dollimore, J. & Kindberg, T. 1995. Distributed Systems, Concepts and Design, 2nd edition. Addison-Wesley Publishing Company. 644 p.

17. ISO. 1995. ITU-T X.901 | ISO/IEC 10746-1 ODP Reference Model Part 1, Overview.

18. OMG. 11.8.2000. Object Management Group Web-site. URL: http://www.omg.org.

19. FAST. 9.10.2000. The FollowMe project page. URL: http://hyperwave.fast.de/ followme.

20. TINA Consortium. 25.8.2000. TINA. URL: http://www.tinac.com/.

21. Lehmann, L., Cadorin, M. & Wuergler C. 1997. Service Creation on a TINA Platform: An Experience Report. Proceedings of the Global Convergence of Telecommunications and Distributed Object Computing (TINA '97).

22. Yarborough, W. 1998. Building Communication Networks with Distributed Objects. McGraw-Hill Companies, Columbus, Ohio, USA. 212 p.

23. Black U. 1995. TCP/IP & Related Protocols, Second Edition. IEEE CS Store. 400 p.

24. Shuey, R., Spooner, D. & Frieder, O. 1997. The Architecture of Distributed Computer Systems: A Data Engineering Perspective on Information Systems. Addison-Wesley Publishing Company. 401 p.

25. ANSA Project. 15.8.1998. Official Record of the ANSA Project. URL: http://www.ansa.co.uk/.

26. Orfali, R. & Harkey, D. 1997. Instant CORBA. John Wiley & Sons. 313 p.

27. Mowbray, T. & Zahavi, R. 1995. The Essential CORBA: Systems Integration Using Distributed Objects. John Wiley & Sons. 336 p.

28. Tari, Z. & Bukhres, O. 2001. Distributed Object Computing: A CORBA Perspective. John Wiley & Sons. 416 p.

29. Oracle Technology Network. 29.8.2000. Oracle Application Server, white paper. URL: http://otn.oracle.com/products/ias/techlisting.html.

30. Microsoft Corp. 26.7.2000. Windows NT Server Terminal Server Edition. . URL: http://www.microsoft.com/ntserver/terminalserver/techdetails/.

31. Citrix Systems Inc. 1.8.2000. Citrix Server-based Computing White Paper. URL: ftp://ftp.citrix.com/doclib/SBCWP.PDF, company info at URL: http://www.citrix.com/.

32. 3GPP. 02.02.2001. Third Generation Partnership Project. URL: http://www.3gpp.org/.

33. ETSI SMG4. 1999. Mobile Station Application Execution Environment (MExE), Specification no. GSM 02.57.

34. WAP Forum. 22.8.2000. Wireless Application Protocol. URL: http://www.wapforum.org/index.htm.

35. ETSI. 22.8.2000. The European Telecommunications Standards Institute. URL: http://www.etsi.org/.

36. OSGi. 23.10.2000. Open Service Gateway initiative. URL: http://www.osgi.org/index.html.

37. Brown, A.W. & Wallnau, K.C. 1998. The Current State of CBSE. IEEE Software, Septemper/October 1998. Pp. 38 - 46.

38. Cetus Team. 11.8.2000. Cetus Links on Objects & Components. URL: http://www.cetus-links.org/.

39. Gopalan, S. R. 23.8.2000. A Detailed Comparison of CORBA, DCOM and Java/RMI. URL: http://www.execpc.com/~gopalan/misc/compare.html.

40. OMG. 23.8.2000. What's Coming in CORBA 3. URL: http://www.omg.org/technology/corba/corba3releaseinfo.htm.

41. Gopalan, S. R. 23.8.2000. A Detailed Comparison of Enterprise JavaBeans (EJB) & The Microsoft Transaction Server (MTS) Models. URL: http://members.tripod.com/gsraj/misc/ejbmts/ejbmtscomp.html.

42. Distributed Systems Technology Centre. 23.8.2000. DCE/CORBA Interworking Specification. URL: http://www.dstc.edu.au/Research/ Projects/ DceCORBA_Interworking/ interworking.html.

43. Java Consortium. 11.8.2000. Java Consortium's Web-site. URL: http://java.sun.com.

44. Orfali, R. & Harkey, D. 1999. Client/Server Survival Guide, 3rd Edition. John Wiley & Sons. 800 p.

45. Edwards, J. 1999. 3-Tier Server/Client at Work, Revised Edition. John Wiley & Sons. 336 p.

46. Wies, R. 1995. Using a Classification of Management Policies for Policy Specification and Policy Transformation, Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management, Santa Barbara, CA, USA, May 1995.

47. Lupu, E. & Sloman, M. 1997. Conflict Analysis for Management Policies. Proceedings of the 5th IFIP/IEEE International Symposium on Integrated Network Management (IM'97).

48. Steenekamp, P.J & Roos, J. 1996. Implementation of Distributed Systems Management Policies: A Framework for the Application of Intelligent Agent Technology, Proceedings of The Second International Workshop on Systems Management, Toronto, Canada", June 1996.

49. Damian, M. & Sloman, M. 1994. Specification of Management Policies. In Proceedings of the Fifth IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM '94), Toulouse, France, October 1994.

50. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. & Lorenson, W. 1991. Object-Oriented Modeling and Design. Prentice Hall. 500 p.

51. Paulk, M., Chrissis, M.B., Weber, C. & Perdue, J. 1997. The Capability Maturity Model for Software, Version 2B. URL: http://www.sei.cmu.edu /pub /cmm /v2 /cmm-v2-slides.pdf.

52. OMG. 29.6.2000. UML v. 1.3 specification. URL: http://www.omg.org/, URL: http://www.rational.com/uml/resources/documentation/.

53. Cheesman, J. & Daniels, J. 2000. UML Components: A Simple Process for Specifying Component-Based Software. Addison-Wesley Publishing Company. 208 p.

54. Schmidt, D.C. & Fayad, M.E, 1997. Lesson learned. Building Reusable OO Frameworks for Distributed Software. Communications of the ACM. Vol. 40, No. 10. Pp. 85 - 87.

55. Bosch, J. 2000. Design and Use of Software Architectures. Addison-Wesley Publishing Company. 400 p.

56. Hofmeister, C., Nord, R. & Son, D. 1999. Applied Software Architecture. Addison-Wesley Publishing Company. 432 p.

57. Bass, L., Clements, P., Kazman, R. & Bass, K. 1998. Software Architecture in Practice (SEI Series in Software Engineering). Addison-Wesley Publishing Company. 452 p.

58. Maier, M. & Rechtin, E. 2000. The Art of Systems Architecting, 2nd Edition. CRC Press. 344 p.

59. Marshall, C. 1999. Enterprise Modeling with UML: Designing Successful Software through Business Analysis. Addison-Wesley. 272 p.

60. Gomaa, H. 2000. Designing Concurrent, Distributed, and Real-Time Applications with UML. Addison-Wesley Publishing Company. 785 p.

61. Kassem, N. 2000. Designing Enterprise Applications with the Java(tm) 2 Platform, Enterprise Edition. Addison-Wesley Publishing Company. 368 p.

62. Lea, D. 1999. Concurrent Programming in Java, Second Edition: Design Principles and Patterns. Addison-Wesley Publishing Company. 432 p.

63. Harold, E.R. 2000. Java Network Programming. O'Reilly & Assoc. 731 p.

64. Taramaa, J. 1998. Practical development of software configuration management for embedded systems. Technical Research Centre of Finland, Espoo. VTT Publications 366. 147 p. + app. 110 p.

65. Weiss, D. & Lai, R. 1999. Software Product-Line Engineering. A family-based Software Development Process. Addison-Wesley. 426 p.

Appendix A: Source code listings

HelloWorldApplication

```
//=======================================================================
//  VTT Electronics                            Java Class Definition File
//
//  Project:    DIT_MMO
//  Task:       Management Framework of Distributed Software Objects
//              and Components.
//  Package:
//  File:       HelloWorldApplication.java
//
//  Version:    v.1.0
//  Date:       07.02.2001
//  Status:     draft       []  10.05.2000 / Markus Moilanen (MMO)
//              proposal    []  11.01.2001 / Markus Moilanen (MMO)
//              accepted    [X] 07.02.2001 / Supervisor of the thesis
//
//
//  Version history:
//
//      Version     Date / Author         Comment
//      ------------------------------------------------------------------
//      0.0         10.05.2000 / MMO      File created
//      0.1         17.05.2000 / MMO      First prototype
//      0.2         19.01.2001 / MMO      Final implementation
//
//  Copyright © 2001, VTT Electronics and the author
//
//
==========================================================================
import fi.vtt.ele.management_framework.implementation.application_layer.*;
import fi.vtt.ele.management_framework.implementation.management_layer.*;
import fi.vtt.ele.management_framework.implementation.utility.*;
/**
 *<TABLE BORDER CELLPADDING=4 CELLSPACING=2 WIDTH=75%>
 *           <TR><TH ALIGN=LEFT  WIDTH=10%>Description</TH>
 *           <TD>
 * This class HelloWorldApplication is a demo application implementing
 * the withPolicy, Representable, and Distributable interfaces.
 * It does not do very much, it can only release the representative proxy
 * and the application associated policy when they are required.
 * Even the demo application can release a multible proxy, but it does not
 * serve them further. To support the multible proxy actions in run time,
 * the communication between proxy and application must be built.
 * oFortunately, excellent binding objects of the solution can be
 * utilised. Extended features can be built on the skeleton of this demo
 * application class.
 *           </TD>
 *           <TR><TH ALIGN=LEFT>Version</TH>
 *      <TD>v1.0</TD>
 *            <TR><TH ALIGN=LEFT>Date</TH>
 *      <TD>07.02.2001</TD>
 *            <TR><TH ALIGN=LEFT>Source</TH>
 *      <TD><A href="HelloWorldApplication.java">
 * HelloWorldApplication.java</A></TD>
 *            <TR><TH ALIGN=LEFT>Author</TH>
 *      <TD>Markus Moilanen (MMO)</TD>
 *</TABLE>
 * @version                v.1.0 07.02.2001
 * @author Markus Moilanen, <A HREF="http://www.iki.fi/~mmoi">
 * www.iki.fi/~mmoi</A>
 */
public class HelloWorldApplication implements withPolicy, Representable,
Distributable
{
  /**
    The method getRepresentantive creates and releases the proxy objects.
    The method can be defined as it is appropriate for every individual
    case; it must only provide the proxyable client object. The operation
    is automatically called by the disribution service.
    @param for_ is the identity of the client requiring the proxy.
    The identity can be used for taking the clients in the account, but
    not to control the clients and their rights in the distributed
    system - the ManagementServer will maintain the rights according to
    the application's policy (included later in this file).
    @return the proxyable client object.
  */
  public Proxyable getRepresentative(Identity for_)
  {
    System.out.println("\nApplication: -The representative proxy has
                          been required by '"+for_.to_String()+"'");
    return new HelloWorldClient();
  }
  /**
    The method getDistributionPolicy releases the distribution policy
    of the application. The policy is defined in this file later on.
    @return the InformationBranch object.
  */
  public InformationBranch getDistributionPolicy()
  {
    HelloWorldApplicationInformationBranch b = new
                            HelloWorldApplicationInformationBranch();
    return (InformationBranch) b;
  }
  /**
    The method run supports the Java threads. When the application is
    installed as a thread, the run method will be called by the start()
    and stop() operations of the environment to maintain the processes
    of the system. When the application is not installed as a thread,
    the run method should be called directly.
  */
  public void run()
  {
    System.out.print("Application: -HelloWorldApplication is running,
                                                press Ctrl+C to quit ..");
    long t = System.currentTimeMillis();
    long wait = 5*1000;
    while (true)
    {
      while (System.currentTimeMillis() <= t+wait) {Thread.yield();}
      t = System.currentTimeMillis();
      System.out.print(".");
    }
  }
/**
  This inner class of the policy definitions of the application is created
  only for the test purpose. This is the place in which the descriptions
  of the policy must be developed as a further study - the definitions
  might be XML-based.
*/
  public class HelloWorldApplicationInformationBranch implements
InformationBranch
  {
    /** The next defin. is only for the demo - will be separated in
        the next version.*/
   public int DIMENSION = Dimension.DISTRIBUTION;
   /** The next definitions are ROOT-related - will be separated in
       the next version. */
    public String _UIN = "HelloWorldApplication";
    /** UID can be produced by the common GUIDGEN-utility. */
    public String _UID = "BE9CD630-A96B-11D3-81BC-00902790ECE5";
    /** The next def. are DISTRIBUTION-related - will be separated in
        the next version.*/
    public int PREFERRED_DOOR = 4777;
    public int MAX_NUMBER_OF_REPRESENTATIVE_OUT = 10;
    public int getDIMENSION() { return DIMENSION; }
    public String getUIN() { return _UIN; }
    public String getUID() { return _UID; }
    public int getPREFERRED_DOOR() { return PREFERRED_DOOR; }
    public int getMAX_NUMBER_OF_REPRESENTATIVE_OUT()
                      { return MAX_NUMBER_OF_REPRESENTATIVE_OUT; }
  }
}
```

HelloWorldClient

```
// ==========================================================================
//  VTT Electronics                              Java Class Definition File
//
//  Project:    DIT_MMO
//  Task:       Management Framework of Distributed Software Objects
//              and Components.
//  Package:
//  File:       HelloWorldClient.java
//
//  Version:    v.1.0
//  Date:       07.02.2001
//  Status:     draft       []  10.05.2000 / Markus Moilanen (MMO)
//              proposal    []  11.01.2001 / Markus Moilanen (MMO)
//              accepted    [X] 07.02.2001 / Supervisor of the thesis
//
//
//  Version history:
//
//      Version     Date / Author         Comment
//      ------------------------------------------------------------------
//      0.0         10.05.2000 / MMO      File created
//      0.1         17.05.2000 / MMO      First prototype
//      0.2         19.01.2001 / MMO      Final implementation
//
//  Copyright © 2001, VTT Electronics and the author
//
// =======================================================================
import
fi.vtt.ele.management_framework.implementation.application_layer.Proxyable
;
import fi.vtt.ele.management_framework.implementation.utility.ReturnCode;
import java.io.Serializable;
import java.io.ObjectOutputStream;
import java.io.ObjectInputStream;
/**
 *<TABLE BORDER CELLPADDING=4 CELLSPACING=2 WIDTH=75%>
 *            <TR><TH ALIGN=LEFT  WIDTH=10%>Description</TH>
 *            <TD>
 * The HelloWorldClient class represents the client part of the
 * HelloWorldApplication class. It implements the Proxyable interface to
 * support the system, but also the Serializable and Cloneable interfaces
 * to support the Java language specific functions to serialise and de-
 * serialise objects. The class contains support to implement those Java
 * specific interfaces directly, but it is convenient to use them directly
 * from Java . A more sophisticated client for a more sophisticated
 * application can be built on this demo client skeleton.
 *            </TD>
 *            <TR><TH ALIGN=LEFT>Version</TH>
 *      <TD>v1.0</TD>
 *            <TR><TH ALIGN=LEFT>Date</TH>
 *      <TD>07.02.2001</TD>
 *            <TR><TH ALIGN=LEFT>Source</TH>
 *      <TD><A href="HelloWorldClient.java">HelloWorldClient.java</A></TD>
 *            <TR><TH ALIGN=LEFT>Author</TH>
 *      <TD>Markus Moilanen (MMO)</TD>
 *</TABLE>
 * @version                v.1.0 07.02.2001
 * @author Markus Moilanen,
 *                 <A HREF="http://www.iki.fi/~mmoi">www.iki.fi/~mmoi</A>
 */
 public class HelloWorldClient implements Proxyable, Serializable,
Cloneable
{
  /**
    This is the default constructor. It must be provided to support
    the Java serialising functions.
  */
```

```
public HelloWorldClient() {};
/**
  The method reConstruct must do everything which is necessory to build
  up the proxyable client object in the target environment. With the
  Java implementation, the serialised object must be cloned because the
  serialising channel owns the object after receiving it and the object
  will be killed with the channel as soon as the associated downloader
  thread is killed by the target system. The method is automatically
  called by the system immediately after the de-serialising of the
  object.
  @return the fully working Proxyable object.
*/
public Proxyable reConstruct()
{
  try
  {
    return (Proxyable)super.clone();
  } catch (Exception e) {return null;}
}
/**
  The method reActivate must be called to activate the run time
  operations which the proxy is intended to do in the target platform,
  e.g. start the associated threads or the communication. The method is
  automatically called by the distribution service on the client side.
  @return true if the object was reactivated successfully.
*/
public boolean reActivate()
{
  System.out.println("Proxyable: -Reactivated.");
  return true;
}
/**
  The method unActivate is an opposite operation to the reActivate. The
  method is automatically called by the distribution service on the
  server side before the serialising of the object.
  @return true if the unactivation was succesful.
*/
public boolean unActivate()
{
  return true;
}
/**
  The method writeProxyable and readProxyable are reserved to be used by
  other platforms than Java. The Java serialise and de-serialise already
  provides the same features.
  @return true if the operation was successful.
*/
public boolean writeProxyable(ObjectOutputStream into_) { return true; }
public boolean readProxyable(ObjectInputStream from_) { return true; }
/**
  The method run supports the Java threads. When the application is
  installed as a thread, the run method will be called by the start()
  and stop() operations of the environment to maintain the processes of
  the system. When the application is not installed as a thread, the run
  method should be called directly.
  */
  public void run()
  {
    System.out.print("Proxyable: -"+getClass().getName()+" is running,
                                            press Ctrl+C to quit ..");
    long t = System.currentTimeMillis();
    long wait = 5*1000;
    while (true)
    {
      while (System.currentTimeMillis() <= t+wait) {Thread.yield();}
      t = System.currentTimeMillis();
      System.out.print(".");
    }
  }
}
```

ManagementServer

```
// ==========================================================================
//  VTT Electronics                            Java Class Definition File
//
//  Project:    DIT_MMO
//  Task:       Management Framework of Distributed Software Objects
//              and Components.
//  Package:
//  File:       ManagementServer.java
//
//  Version:    v.1.0
//  Date:       07.02.2001
//  Status:     draft       []  10.05.2000 / Markus Moilanen (MMO)
//              proposal    []  11.01.2001 / Markus Moilanen (MMO)
//              accepted    [X] 07.02.2001 / Supervisor of the thesis
//
//  Version history:
//
//      Version     Date / Author         Comment
//      ----------------------------------------------------------------
//      0.0         10.05.2000 / MMO      File created
//      0.1         17.05.2000 / MMO      First prototype
//      0.2         19.01.2001 / MMO      Final implementation
//      0.3         05.02.2001 / MMO      The code is finished
//
//  Copyright © 2001, VTT Electronics and the author
//
//===========================================================================
import fi.vtt.ele.management_framework.implementation.management_layer.*;
import fi.vtt.ele.management_framework.implementation.application_layer.*;
import fi.vtt.ele.management_framework.implementation.utility.*;
import java.io.*;
/**
 *<TABLE BORDER CELLPADDING=4 CELLSPACING=2 WIDTH=75%>
 *            <TR><TH ALIGN=LEFT  WIDTH=10%>Description</TH>
 *            <TD>
 * The ManagementServer class implements the Management interface. The
 * class offers all the necessary methods to maintain the policy that the
 * application expresses. The name of the application is passed as
 * a parameter. The application's policy (if exists) will be interpreted
 * by the ManagementServerand associated services will be started
 * accordingly if possible. The class is descripted thoroughly
 * in Chapter 5, Subsection 5.2.1 of the thesis.
 *          </TD>
 *          <TR><TH ALIGN=LEFT>Version</TH>
 *      <TD>v1.0</TD>
 *          <TR><TH ALIGN=LEFT>Date</TH>
 *      <TD>07.02.2001</TD>
 *          <TR><TH ALIGN=LEFT>Source</TH>
 *      <TD><A href="ManagementServer.java">ManagementServer.java</A></TD>
 *          <TR><TH ALIGN=LEFT>Author</TH>
 *      <TD>Markus Moilanen (MMO)</TD>
 *</TABLE>
 * @version                v.1.0 07.02.2001
 * @author Markus Moilanen, <A
 *                      HREF="http://www.iki.fi/~mmoi">www.iki.fi/~mmoi</A>
 */
class ManagementServer implements Management, Runnable
{
  public withPolicy application_ = null;
  public Decision decision_table[][] = new Decision[5][5];
  public Dimension _to_support[] = new Dimension[5];
  public Policy _on_demand[] = new Policy[5];
  public ReturnCode _on_init = null;
  public ReturnCode _on_decoding = null;
  public ReturnCode _on_policy = null;
  public ReturnCode _on_idle = null;
  public int _dimension = 0;
  private String _UIN_ = null;
  private String signature = null;
  public Policy management_policy = null;
  private int state = INITIALISING;
  private int sub_state = 0;
  /**
    This constructor is called by a End-user Terminal object. In the demo,
    the End-user Terminal object is a usual command window of the system.
    The command window shares the same JVM (Java Virtual Machine) with
    other command windows.  If something goes wrong, the constructor
    exits.
    @param _from_UIN_ is the application name to be started.
  */
  public ManagementServer(String _from_UIN_) throws Exception
  {
    _UIN_ = _from_UIN_;
    state=INITIALISING;
    _on_init = createApplication(_from_UIN_);
    if (_on_init.isFAILED()) {exit(_on_init.getString());}
      else if (_on_init.isINFOCODE()) {exit(_on_init.getString());}
        else if (_on_init.isSUCCESS())
        {
          for (int i=0;i<=4;i++) _on_demand[i]=null;
          state=DECODING_POLICY;
          _on_decoding = decodePolicies(application_);
          if (_on_decoding.isSUCCESS())
          {
            for (int i=0;i<=4;i++) _to_support[i]=null;
            _dimension=0;
            state = CREATING_DIMENSIONS;
            for (_dimension=0; _dimension <= 4; _dimension++)
            {
              if (_on_demand[_dimension]!=null)
              {
                _to_support[_dimension] =
                          createDimensionServer(_on_demand[_dimension]);
              }
            }
            for (int i=0;i<=4;i++) for (int j=0;j<=4;j++)
                                              decision_table[i][j]=null;
            state=POLICYMAKING;
            _on_policy = policyMaking();
            printDecisiontable();
            if (_on_policy.isSUCCESS())
            {
              application_.run();
              state=IDLE;
            } else if (_on_policy.isFAILED())
                                          exit(_on_policy.getString());
                else exit(_on_policy.getString());
          } else if (_on_decoding.isFAILED())
                                      {exit(_on_decoding.getString());}
              else exit(_on_decoding.getString());
        }
  }
  /**
    A private method printDecisiontable prints the decision table
    to illustrate the results of the method policyMaking.
  */
  private void printDecisiontable()
  {
    boolean s = false;
    System.out.println
("---------------------------------------------------------------------");
    System.out.println("ManagementServer: -Decision table is resolved as
follows (see Table 15):");
    System.out.println
("---------------------------------------------------------------------");
    System.out.println("\t \t \tAdv.\tDis.\tRes.\tSec.\tPro.\t");
    System.out.print("\t \t \t");
    for (int i=0;i<=4;i++)
    System.out.print(decision_table[i][0].DEMANDS()+"\t");
    System.out.print("\n");
    System.out.print("Advertisement\t");
    for (int i=0;i<=4;i++) if (decision_table[i][0].SUPPORTS())
                                                             {s = true;}
    if (s) System.out.print("true\t");  else System.out.print("false\t");
    for (int i=0;i<=4;i++) if (decision_table[i][0].JUDGMENT())
               {System.out.print("true\t");}else System.out.print(" \t");
    System.out.print("\n");s = false;
    System.out.print("Distribution\t");
    for (int i=0;i<=4;i++) if (decision_table[i][1].SUPPORTS())
                                                          {s = true;}
    if (s) System.out.print("true\t");  else System.out.print("false\t");
    for (int i=0;i<=4;i++) if (decision_table[i][1].JUDGMENT())
              {System.out.print("true\t");} else System.out.print(" \t");
    System.out.print("\n");s = false;
    System.out.print("Resource manag.\t");
    for (int i=0;i<=4;i++) if (decision_table[i][2].SUPPORTS())
                                                          {s = true;}
    if (s) System.out.print("true\t");  else System.out.print("false\t");
    for (int i=0;i<=4;i++) if (decision_table[i][2].JUDGMENT())
              {System.out.print("true\t");} else System.out.print(" \t");
    System.out.print("\n");
    System.out.print("Security \t");s = false;
    for (int i=0;i<=4;i++) if (decision_table[i][3].SUPPORTS())
                                                          {s = true;}
    if (s) System.out.print("true\t");  else System.out.print("false\t");
    for (int i=0;i<=4;i++) if (decision_table[i][3].JUDGMENT())
              {System.out.print("true\t");} else System.out.print(" \t");
    System.out.print("\n");
    System.out.print("Profile manag.\t"); s = false;
    for (int i=0;i<=4;i++) if (decision_table[i][4].SUPPORTS())
                                                          {s = true;}
    if (s) System.out.print("true\t");  else System.out.print("false\t");
    for (int i=0;i<=4;i++) if (decision_table[i][4].JUDGMENT())
              {System.out.print("true\t");} else System.out.print(" \t");
    System.out.print("\n");
    System.out.println
("----------------------------------------------------------------------");
  }
  /**
    The policyMaking creates the policy to be followed during the
    ManagementServer object runs. It considers between the demands of the
    application and the possibilities of the local platform to carry the
    demanded features. The method will be developed further in
    the future versions.
    @return the ReturnCode object with SUCCESS if the operation
    was successful, otherwise returns with FAILED.
  */
  public ReturnCode policyMaking()
  {
    ReturnCode ret = getReturnCode();
    try
    {
      for (int i=0;i<=4;i++)
      {
      for (int j=0;j<=4;j++)
      {
        if ((_on_demand[i]==null) && (_to_support[j] == null))
  decision_table[i][j] = new DecisionImplementation(false, false, false);
        if ((_on_demand[i]!=null) && (_to_support[j] == null))
  decision_table[i][j] = new DecisionImplementation(true, false, false);
        if ((_on_demand[i]==null) && (_to_support[j] != null))
  decision_table[i][j] = new DecisionImplementation(false, true, false);
        if ((_on_demand[i]!=null) && (_to_support[j] != null))
  decision_table[i][j] = new DecisionImplementation(true, true, true);
      }
      }
      ret.setSUCCESS();
    } catch (Exception e) {ret.setFAILED(); ret.setBody
                           ("ManagementServer: -Policymaking failed.");}
    return ret;
  }
  /**
    The method decodePolicies decodes the policy which the application
    expresses by calling the method createDimensionPolicy. First it will
    try to cast the given application object to implement the different
    interfaces of the dimensions. If the casting operation is successful,
    it will try to reach the branch policy by calling the appropriate,
    known operation of the interface to reach the policy of the
    application object. This is an unstable operation which may fail as
    well as be successful depending on the application and the policies
    which it implements. Successfully defined branch policies will be put
    in the on_demand table from where they will be easily reached later.
    @return the ReturnCode object with SUCCESS if the decoding was
    successful, otherwise returs with FAILED.
  */
  private ReturnCode decodePolicies(withPolicy appl_)
  {
    ReturnCode ret = getReturnCode();
    ret.setFAILED();
    try
    {
      management_policy = createManagementPolicy(null);
      Distributable d = (Distributable)appl_;
      InformationBranch p = d.getDistributionPolicy();
      int dim = p.getDIMENSION();
      _on_demand[getIndex(dim)] =
                      createDimensionPolicy(d.getDistributionPolicy());
      ret.setSUCCESS();
    } catch (Exception e) {}
    return ret;
  }
  /**
    The private method getIndex takes the "binary logarithm" from
    the constant which expresses the dimension (see the interface
    Dimension). This is needed when the dimensions are handled according
    to the policy branches of the corresponding dimensions.
    @param dim_ is a constant representing the dimension.
  */
  private int getIndex(int dim_) throws Exception
  {
    if ((dim_ > Dimension.PROFILE_MANAGEMENT)) throw
           new Exception("ManagementServer: -Dimension does not exist.");
    else
    {
      switch (dim_)
      {
        case 1:  return 0;
        case 2:  return 1;
        case 4:  return 2;
        case 8:  return 3;
        case 16: return 4;
       default: throw new Exception
                        ("ManagementServer: -Dimension does not exist.");
      }
    }
  }
  /**
    The method createDimensionPolicy creates the dimension policy object.
    In the future version, the policy will be parsed from the XML-source.
    This area is included in the issues of the further study.
    @param _from_ is the InformationBranch object by which the policy
    must be formed.
    @return the Policy object.
  */
  public Policy createDimensionPolicy(InformationBranch _from_)
  {
    try
    {
      Policy p = new ManagementPolicy(_from_);
      return p;
    } catch (Exception e) {return null;}
  }
  /**
    The method createDimensionServer creates the dimensions services
    according to the policy of the dimension. In this version, only
    the DistributionDimensionServer can be created, others return null.
    @param _by_ is the ploicy object by which the dimension must be
    installed.
    @return the Dimension object.
  */
  public Dimension createDimensionServer(Policy _by_)
  {
    if (_by_.getDIMENSION() == Dimension.ADVERTISEMENT) {return null;}
    else
    if (_by_.getDIMENSION() == Dimension.DISTRIBUTION)
    {
      return new DistributionDimensionServer(this, _by_);
    }
    if (_by_.getDIMENSION() == Dimension.RESOURCE_MANAGEMENT)
                                                         {return null;}
    else
    if (_by_.getDIMENSION() == Dimension.SECURITY) {return null;}
    else return null;
  }
  /**
    The method createApplication creates and installs the application
    object if the application can express the policy. The method is
    explained thorouhgly in Chapter 5, Subsection 5.2.1 of the thesis.
    @param _from_UIN_ is the unique name of the application.
    @return the ReturnCode with SUCCESS if operation was successful,
    otherwise returns with FAILED.
```

```
*/
public ReturnCode createApplication(String _from_UIN_)
{
  ReturnCode ret = getReturnCode();
  Class check_ = null;
  try
  {
    File f = new File(_from_UIN_);
  } catch (NullPointerException e)
  { /** An object is not located */
    ret.setFAILED();
    ret.setBody(e.toString());
    return ret;
  }
  try
  {
    check_ = Class.forName(_from_UIN_);
  } catch (ClassNotFoundException f)
  { /** An object is not identified */
    ret.setFAILED();
    ret.setBody(f.toString());
    return (ReturnCode) ret;
  }
 /** An object is located and identified, but is it a one withPolicy? */
  try
  {
    /** Getting signature but not handling it in this version */
    application_ = (withPolicy)check_.newInstance();
    signature = application_.POLICY_SIGNATURE;
    ret.setSUCCESS();
  } catch (Exception g)
  {  /** An object is not withPolicy */
    ret.setINFOCODE();
    ret.setBody(g.toString());
  }
  return (ReturnCode) ret;
}

public Policy createManagementPolicy(InformationRoot _from_)
{
  // parameter _from_ not used in this version
  try
  {
    return new ManagementPolicy();
  } catch (Exception e) {return null;}
}
/**
  The method handleManagementInfo handles the management information
  coming from different dimension services. In the demo version it
  grants all the operations, but in the further version it will maintain
  the dimension policies by some generic handlers to be developed.
  @param _from_ is a ManagementAction object coming from the calling
  dimension object.
  @return the ReturnCode object with SUCCESS if operation was
  successful, otherwise returns FAILED.
  */
  public ReturnCode handleManagementInfo(ManagementAction _from_)
  {
    /** We will grant them all in the demo.*/
    _from_.setGRANTED();
    _from_.setSUCCESS();
    return _from_.createReturnCode();
  }
  /**
    The method commitManagementAction will proceed the operation needed
    to realise the policy on the Identity object. In the demo version,
    it does nothing.
  */
  public ReturnCode commitManagementAction(Identity _on_, ManagementAction
_with_)
  {
    return _with_.createReturnCode();
  }
  public withPolicy getApplication() {return application_;}
  public Policy getManagementPolicy() {return management_policy;}
  public void exit(String _with_)
  {
    System.out.println(_with_);
    System.exit(0);
  }

  public void run() { }

  public ReturnCode getReturnCode()
  {
    ReturnCode ret = null;
    try { return new  ReturnCodeImplementation(); }
    catch (Exception e) {exit(e.toString());}
    return ret;
  }
  /** Main-function for start the testing. */
  public static void main(String[] args) throws Exception
  {
    System.setSecurityManager(null);
    ManagementServer server = new  ManagementServer(args[0]);
  }
/*************************************************************************
  This inner class ManagementPolicy is created only for the test purpose.
  This is the place in where the descriptions of the policy must be
  developed as a further study - might be XML-based.
*************************************************************************/
  public class ManagementPolicy implements Policy
  { /** The class is presented in the Subsection 6.3.3. */ }
```

DistributionDimensionServer

```
//========================================================================
//  VTT Electronics                             Java Class Definition File
//
```

```
//  Project:    DIT_MMO
//  Task:       Management Framework of Distributed Software Objects
//              and Components.
//  Package:
//  File:       DistributionDimensionServer.java
//  Version:    v.1.0
//  Date:       07.02.2001
//  Status:     draft       []  10.05.2000 / Markus Moilanen (MMO)
//              proposal    []  11.01.2001 / Markus Moilanen (MMO)
//              accepted    [X] 07.02.2001 / Supervisor of the thesis
//
//  Version history:
//
//      Version     Date / Author         Comment
//      ------------------------------------------------------------------
//      0.0         10.05.2000 / MMO      File created
//      0.1         17.05.2000 / MMO      First prototype
//      0.2         19.01.2001 / MMO      Final implementation
//      0.3         05.02.2001 / MMO      Program is finished
//
//  Copyright © 2001, VTT Electronics and the author
//========================================================================
import fi.vtt.ele.management_framework.implementation.management_layer.*;
import fi.vtt.ele.management_framework.implementation.application_layer.*;
import
fi.vtt.ele.management_framework.implementation.distribution_transparency.*
;
import fi.vtt.ele.management_framework.implementation.utility.*;
/**
 *<TABLE BORDER CELLPADDING=4 CELLSPACING=2 WIDTH=75%>
 *           <TR><TH ALIGN=LEFT  WIDTH=10%>Description</TH>
 *           <TD>
 * The class DistributionDimensionServer implements the Distribution and
 * Dimension interfaces. All the methods are implemented the way the
 * mentioned interfaces demand.
 *           </TD>
 *           <TR><TH ALIGN=LEFT>Version</TH>
 *      <TD>v1.0</TD>
 *           <TR><TH ALIGN=LEFT>Date</TH>
 *      <TD>07.02.2001</TD>
 *           <TR><TH ALIGN=LEFT>Source</TH>
 *      <TD><A href="DistributionDimensionServer.java">
 * DistributionDimensionServer.java</A></TD>
 *           <TR><TH ALIGN=LEFT>Author</TH>
 *      <TD>Markus Moilanen (MMO)</TD>
 *</TABLE>
 * @version                v.1.0 07.02.2001
 * @author Markus Moilanen,
 * <A HREF="http://www.iki.fi/~mmoi">www.iki.fi/~mmoi</A>
 */
public class DistributionDimensionServer implements Distribution,
Dimension
{
  private Management management_server = null;
  private Policy distribution_policy = null;
  private Proxyable proxy = null;
  private int state = INITIALISING;
  /** Identity of the application which provides the service. */
  private Identity service = null;
  private NetworkTransparency bo = null;
  private Messager messager = null;
  private ObjectUploader ou =  null;
  /**
    This constructor will initialise the disribution dimension server.
    @param server_ is the associated management server object representing
    the owner of this distribution server object.
    @param distribution policy_ is the policy object by which the
    distribution service must be maintained.
  */
  public DistributionDimensionServer(Management server_, Policy
distribution_policy_)
  {
    management_server = server_;
    distribution_policy = distribution_policy_;
    try
    {
      Address service_addr = new AddressImplementation
              (            null,distribution_policy_.getPREFERRED_DOOR());
      service = new IdentityImplementation(distribution_policy_.getUIN());
      bo = new TCPIPNetworkTransparencyBO();      System.out.println
("**************************************************************");
      System.out.println("Demo: Name Cache will be configured for the
                                                      demo according to");
      System.out.println("the disribution policy that the application
                                                           expresses ..");
      bo.setAddress(service, service_addr);
      bo.listNameCache();
      bo.saveNameCache("democache.txt");
      System.out.println(".. and the entries will be saved in the
                                             cachefile 'democache.txt'.");
      System.out.println("The file may be disributed as a replacement of
                                                   a directory service.");
System.out.println
("**************************************************************");
      if (initMessager(service,bo))
      {
        startListen();
        state = IDLING;
        System.out.println("Distribution: -Accepting clients ..");
      } else exit("Distribution: -Service could not be started.");
    } catch (Exception e) {exit("Distribution: -Service could not
                                                   be initialised.");}
  }
  /**
    The initMessager method installs the Messager utility object.
    @param _for_ is the identity of the application object.
    @param bo_ is the binding object to use.
    @return true if the messager was initialised successfully.
  */
  public boolean initMessager(Identity _for_, NetworkTransparency bo_)
```

```
{
  try
  {
    messager = new Messager(_for_, bo_, this);
  } catch (Exception e) {return false;}
  return true;
}
/**
  The startListen method installs the Messager utility object installed
  in the method initMessager as a thread.
  @return true if a thread was initialised successfully.
*/
public boolean startListen()
{
  Thread mt = new Thread(messager);
  mt.start();
  return true;
}
/**
  The dispatchMessage is the operation which the Messager object will
  call when a message is received. It must pass the message to the
  message associated identity.
  @param message_ the received message.
  @return true if the message was handled here, false if the message
  must be handled by some other dispatcher.
*/
public boolean dispatchMessage(Message message_)
{
  ManagementAction access_entry = null;
  Identity temp_upload_identity = null;
  try
  {
    access_entry = new ManagementActionImplementation();
    access_entry.adaptMessage(message_);
    access_entry.setFAILED();

    if ((access_entry.isQUERY())
                   && access_entry.isCLASS_BYTE_CODE_RESOURCE())
    {
      if (management_server != null)
      {
        withPolicy wp = management_server.getApplication();
        Representable da = (Representable)wp;
        proxy = da.getRepresentative(access_entry.getFromIdentity());
        String resname = proxy.getClass().getName()+".class";
        DiskFile res = new DiskFile(resname);
        access_entry.setObject(res);
        temp_upload_identity = new IdentityImplementation
         (service.to_String()+"_resourcing_for_"+
                         message_.getFromIdentity().to_String());
        access_entry.setToIdentity(temp_upload_identity);
        if (management_server.handleManagementInfo
                             (access_entry).isSUCCESS())
        {
          if (acceptService(access_entry).isSUCCESS())
          {
            access_entry.swapIdentities();
            messager.sendMessage(access_entry.createMessage());
          }
          return true;
        }
      } else exit("Distribution:
                            -Class byte code resource is not found.");
    }
    if ((access_entry.isQUERY())&&access_entry.isREPRESENTATIVE_PROXY())
    {
      if (management_server != null)
      {
        if (proxy == null)
        {
          withPolicy wp = management_server.getApplication();
          Representable da = (Representable)wp;
          proxy = da.getRepresentative(access_entry.getFromIdentity());
        }
        access_entry.setObject(proxy);
        temp_upload_identity = new IdentityImplementation
                  (service.to_String() +"_distribution_for_"
                       + message_.getFromIdentity().to_String());
        access_entry.setToIdentity(temp_upload_identity);
        if (management_server.handleManagementInfo
                                            (access_entry).isSUCCESS())
        {
          if (acceptService(access_entry).isSUCCESS())
          {
            access_entry.swapIdentities();
            messager.sendMessage(access_entry.createMessage());
          }
          return true;
        }
      } else exit("Distribution: -Representative proxy is not found.");
    }
  } catch (Exception e) {return true;}
  return false;
}
/**
  The sendMessage method sends message to the identity coded in
  the ReturnCode class. It adds the client identity in the
  message to be send to represent the sender.
  @param _to_ is the identity to whom the message is to be send.
  @param returncode_ contains the message's contents.
  @return true if the sending was successful, otherwise returns false.
*/
public boolean sendMessage(Identity _to_, ReturnCode returncode_)
{
  try
  {
    Message msg = new MessageImplementation(service, _to_, returncode_);
    messager.sendMessage(msg);
    return true;
   } catch (Exception e) {return false;}
```

```
}
/**
  The handleAction method is the operation of which the ObjectDownloader
  and the ObjectUploader will call when they have completed their tasks.
  @param _action_ is the ManagementAction object which a loader has
  got as an instruction when it have been initialised.
  @return true if the action was handled here, else returns false
  meaning that the action must be handled by some other action handler.
*/
public boolean handleAction(ManagementAction _action_)
{
  if (_action_.isSUCCESS())
  {
    System.out.println("Distribution: -The client '"
                            +_action_.getToIdentity().to_String()
    +"' is served successfully with the reply '"
      +_action_.createReturnCode().to_String()+"'.");
    return true; // returncode is handled
  } else
  {
    System.out.println("Distribution: -An action on the client '"
    +_action_.getToIdentity().to_String()+"' is failed with the reply '"
    +_action_.createReturnCode().to_String()+"'.");
    return true;
  }
}
/**
  The reachService method reaches the service given by the service
  identity. This method is deployed by a ManagementClient object.
  @param _of_ is the identity of the application object offering
  a service.
  @return ReturnCode object.
*/
public ReturnCode reachService(Identity _of_) { return null; }
/**
  The acceptService method starts the ObjectUploader objects to
  be executed as a thread.
  @param _entry_ is a ManagementAction object by which the upload
  operation must be proceeded.
  @return ReturnCode object.
*/
public ReturnCode acceptService(ManagementAction _entry_)
{
  _entry_.setFAILED();
  try
  {
    Thread t = new Thread(new ObjectUploader(_entry_, bo, this));
    t.start();
    _entry_.setSUCCESS();
  } catch (Exception e) {return _entry_.createReturnCode();}
  return _entry_.createReturnCode();
}
public void exit(String exit_msg_)
{
  System.out.println(exit_msg_);
```

```
    System.exit(1);
  }
}
```

ManagementClient

```
//===========================================================================
//  VTT Electronics                             Java Class Definition File
//
//  Project:    DIT_MMO
//  Task:       Management Framework of Distributed Software Objects
//              and Components.
//  Package:
//  File:       ManagementClient.java
//
//  Version:    v.1.0
//  Date:       07.02.2001
//  Status:     draft       []  10.05.2000 / Markus Moilanen (MMO)
//              proposal    []  11.01.2001 / Markus Moilanen (MMO)
//              accepted    [X] 07.02.2001 / Supervisor of the thesis
//
//  Version history:
//
//      Version     Date / Author         Comment
//      -----------------------------------------------------------------
------
//      0.0         10.05.2000 / MMO      File created
//      0.1         17.05.2000 / MMO      First prototype
//      0.2         19.01.2001 / MMO      Final implementation
//      0.3         05.02.2001 / MMO      The code is finished
//
//  Copyright © 2001, VTT Electronics and the author
//
//===========================================================================
import fi.vtt.ele.management_framework.implementation.management_layer.*;
import fi.vtt.ele.management_framework.implementation.application_layer.*;
import fi.vtt.ele.management_framework.implementation.utility.*;
import
fi.vtt.ele.management_framework.implementation.distribution_transparency.*
;
/**
 *<TABLE BORDER CELLPADDING=4 CELLSPACING=2 WIDTH=75%>
 *            <TR><TH ALIGN=LEFT  WIDTH=10%>Description</TH>
 *            <TD>
 * The ManagementClient class implements the Distribution and Dimension
 * interfaces and thus can reach the ability to communicate and download
 * objects. The ManagementClient may implement all the dimensions at once
 * sor only the selected set of the dimensions. The realisation depends on
 * the possibilities which the target environment has, or which dimensions
 * have been selected to be enough for every individual target platform.
 *            </TD>
 *            <TR><TH ALIGN=LEFT>Version</TH>
 *      <TD>v1.0</TD>
 *            <TR><TH ALIGN=LEFT>Date</TH>
 *      <TD>07.02.2001</TD>
 *           <TR><TH ALIGN=LEFT>Source</TH>
 *      <TD><A href="ManagementClient.java">ManagementClient.java</A></TD>
 *           <TR><TH ALIGN=LEFT>Author</TH>
 *      <TD>Markus Moilanen (MMO)</TD>
 *</TABLE>
 * @version                v.1.0 07.02.2001
 * @author Markus Moilanen,
 * <A HREF="http://www.iki.fi/~mmoi">www.iki.fi/~mmoi</A>
 */
public class ManagementClient implements Distribution, Dimension
{
  public Proxyable reconstructed = null;
  private int state = INITIALISING;
  private int REACHING_RESOURCE = 1000;
  private int dimensions = ROOT+DISTRIBUTION;
  private Identity client = null;
  private Identity service = null;
  private NetworkTransparency bo = null;
  private Messager messager = null;
  /**
    This constructor initialises the ManagementClient class. End-user
    Terminal passes the identity of the service to be reached and the
    identity of the current user. The demo version needs the local name
    cache to be configured to contain the actual physical address of the
    service. The binding object will use the cache to resolve the address
    of the service identity.
    @param service_ is the identity of the service to be reached.
    @param clients_ is the identity of the current user.
  */
  public ManagementClient(Identity service_, Identity client_)
  {
    service = service_;
    client = client_;
    try
    {
      bo = new TCPIPNetworkTransparencyBO();
      System.out.println
("************************************************************");
System.out.println
("Demo: Name Cache will be configured for the demo in using   ");
System.out.println
("the default cachefile 'democache.txt' instead of the use of ");
System.out.println
("a directory service - directory services are not available. ");
      bo.loadNameCache("democache.txt");
      bo.listNameCache();
      bo.setAddress(client_, new AddressImplementation(null,0));
      System.out.println
("**********************************************************");
      if (initMessager(client_, bo))
      {
        state = REACHING_RESOURCE;
        if (reachService(service_).isSUCCESS())
        {
```

```
        while (state != REACHING_PROXYABLE) {Thread.yield();}
      }
      if (reachService(service_).isSUCCESS())
      {
        while (state != RUNNING_PROXYABLE) {Thread.yield();}
        reconstructed.run();
      }
    } else exit("ManagementClient: -The service '"
        +service.to_String()+"' could not be reached.");
  } catch (Exception e) {exit
            ("ManagementClient: -Could not be installed.");}
}
/**
  The initMessager method installs the Messager utility object.
  @param _for_ is the identity of the client object.
  @param bo_ is the binding object to use.
  @return true if the messager was initialised successfully.
*/
public boolean initMessager(Identity _for_, NetworkTransparency bo_)
{
  try
  {
    messager = new Messager(_for_, bo_, null);
    return true;
  } catch (Exception e) { return false; }
}
/**
  The startListen method installs the Messager utility object installed
  in the method initMessager as a thread. Not used by this
  ManagementClient.
  @return true if a thread was initialised successfully.
*/
public boolean startListen() {return true;}
/**
  The dispatchMessage is the operation which the messager object will
  call when a message is received. It must pass the message to the
  message associated identity. Not used by this ManagementClient.
  @param message_ the received message. Not used by this
  ManagementClient.
  @return true if the message was handled here, false if the message
  must be handled by some other dispatcher.
*/
public boolean dispatchMessage(Message message_) { return false; }
/**
  The handleAction method is the operation which the ObjectDownloader
  and the ObjectUploader will call when they have completed their tasks.
  @param _action_ is the ManagementAction object that a loader has
  received as an instruction when it has been initialised.
  @return true if the action was handled here, else returns false
  meaning that the action must be handled by some other action handler.
*/
public boolean handleAction(ManagementAction _action_)
{
  if (state == DOWNLOADING_OBJECT)
  {
    if (_action_.isSUCCESS()) System.out.println
           ("ManagementClient: -ObjectDownloader informs that
                         the operation was proceeded successfully.");
    if ((_action_.isSUCCESS())&&(_action_.isREPRESENTATIVE_PROXY()))
    {
      Proxyable p = (Proxyable)_action_.getObject();
      reconstructed = p.reConstruct();
      if (reconstructed == null) exit
   ("ManagementClient: -Representavive proxy cannot be reconstructed.");
      state = REACTIVATING_PROXYABLE;
      if (reconstructed.reActivate())
      {
        state = RUNNING_PROXYABLE;
      } else exit
     ("ManagementClient: -Representavive proxy cannot be reactivated.");
    } else
    if ((_action_.isSUCCESS())&&(_action_.isCLASS_BYTE_CODE_RESOURCE()))
    {
      DiskFile p = (DiskFile)_action_.getObject();
      if (!p.toDisk()) exit
   ("ManagementClient: -Class byte code resource cannot be installed.");
      state = REACHING_PROXYABLE;
    } else exit("ManagementClient: -Object cannot be downloaded.");
    return true;
  } else { return false;}
}
/**
  The reachService method reaches the service given by
  the service identity.
  @param _of_ is the identity of the application object offering
  a service.
  @return ReturnCode object.
*/
public ReturnCode reachService(Identity _of_)
{
  ReturnCode sendquery = null;
  ManagementAction action = null;
  Message coming = null;
  try
  {
    sendquery = new ReturnCodeImplementation();
  } catch (Exception e) {exit
                      ("ManagementClient: -Could not create object.");}
  if (state == REACHING_PROXYABLE)
  {
    sendquery.setQUERY();
    sendquery.setREPRESENTATIVE_PROXY();
    sendquery.setINFOCODE();
    if (sendMessage(_of_, sendquery))
    {
      messager.setTIMEOUT(10000);
      System.out.println("ManagementClient: -Reaching for the service's
                               proxy for no longer than 10 seconds ..");
      coming = messager.receiveMessage();
      messager.setTIMEOUT(0);
      if (coming.isGRANTED())
      {
        System.out.println("ManagementClient: -Representative proxy is
                                  granted, starting the downloader ..");
        try
        {
          action = new ManagementActionImplementation();
          action.adaptMessage(coming);
          Thread t = new Thread(new ObjectDownloader(action, bo, this));
          state = DOWNLOADING_OBJECT;
          t.start();
          action.setSUCCESS();
          return action.createReturnCode();
        } catch (Exception e) {return action.createReturnCode();}
      }
    } else if (coming.isDENIED()) { exit( "ManagementClient:
            -Representative proxy is denied by the server, quitting.");}
    else { exit("ManagementClient: -Cannot send a message.");}
  }
  if (state == REACHING_RESOURCE)
  {
    sendquery.setQUERY();
    sendquery.setCLASS_BYTE_CODE_RESOURCE();
    sendquery.setINFOCODE();
    if (sendMessage(_of_, sendquery))
    {
      messager.setTIMEOUT(10000);
      System.out.println("ManagementClient: -Reaching for the class byte
                      code resource for no longer than 10 seconds ..");
      coming = messager.receiveMessage();
      messager.setTIMEOUT(0);
      if (coming.isGRANTED())
      {
        System.out.println("ManagementClient:
     -Class byte code resource is granted, starting the downloader ..");
        try
        {
          action = new ManagementActionImplementation();
          action.adaptMessage(coming);
          Thread t = new Thread(new ObjectDownloader(action, bo, this));
          state = DOWNLOADING_OBJECT;
          t.start();
          action.setSUCCESS();
          return action.createReturnCode();
        } catch (Exception e) {return action.createReturnCode();}
      }
    } else if (coming.isDENIED()) { exit("ManagementClient:
  -Class byte code resource is denied by the server, quitting.");}
    else { exit("ManagementClient: -Cannot send message.");}
  }
  return action.createReturnCode();
}
/**
  The sendMessage method sends a message to the identity coded in
  the ReturnCode class. It adds the client identity in the
```

```
    message to be sent to represent the sender.
    @param _to_ is the identity to whom the message is to be sent.
    @param returncode_ contains the message's contents.
    @return true if the sending was successful, otherwise returns false.
  */
  public boolean sendMessage(Identity _to_, ReturnCode returncode_)
  {
    try
    {
      Message msg = new MessageImplementation(client, _to_, returncode_);
      messager.sendMessage(msg);
    } catch (Exception e) {return false;}
    return true;
  }
  /**
    The acceptService method starts the ObjectUploader objects to
    be executed as a thread. Not used by this ManagementClient.
    @param _entry_ is a ManagementAction object with which the upload
    operation must be proceeded.
    @return ReturnCode object.
  */
  public ReturnCode acceptService(ManagementAction access_entry_)
                                                         {return null;}
  public void exit(String exit_msg_)
  {
    System.out.println(exit_msg_);

System.out.println("************************************************");
System.out.println("Demo: Name Cache at the end of the execution is:");
    try
    {
      bo.listNameCache();
    } catch (Exception e)
    {
      System.out.println("        Name Cache is not standing.");
    }

System.out.println("************************************************");
System.exit(1);
  }
  /** Main-function is for start the class in command line. */
  public static void main(String[] args)
  {
    System.setSecurityManager(null);
    String usage_str =
 "Usage: 'java ManagementClient <service_UIN> <client_UIN>, where the"
+"<service_UIN> is the service's unique name, e.g. HelloWorldApplication "
+"and the <client_UIN> is the user's unique (and friendly) name, e.g. "
+"Marcus. The expression of the identity will be expanded in the future "
+"version also to carry the less-friendly expression like 'BE9CD630-A96B-"
+"11D3-81BC-00902790ECE5' to distinguish identities properly."
+"Before use, the name cache must be configured accordingly to express the
"
+"physical address of the service identities - usually this is provided  "
+"by a directory service in a distributed system. By default, the name "
+"cache will be configured to carry both addresses in the local host. "
+" You must start the application <service_UIN> with management before "
+" the use of this client - it will configure the name cache for the "
+"service. If the client is used through another subdirectory of the "
+"computer disk, the name cache file must be copied to the directory in "
+"which the ManagementClient is installed after the service has  "
+"configured the file. This is simple because we do not have any "
+"directory service available for the demo.";
    if (args.length == 2)
    {
      if ((args[0]!=null)&&(args[1]!=null))
      {
        try
        {
          Identity service_identity = new IdentityImplementation(args[0]);
          Identity client_identity = new IdentityImplementation( args[1]);
          ManagementClient m = new ManagementClient
                                        (service_identity, client_identity);
        } catch (Exception e) {System.out.println(e.toString());}
      }
    }
    else System.out.println(usage_str);
  }
}
```

TCPIPNetworkTransparencyBindingObject

```
//==========================================================================
//  VTT Electronics                               Java Class Definition File
//
//  Project:    DIT_MMO
//  Task:       Management Framework of Distributed Software Objects
//              and Components.
//  Package:
//  File:       TCPIPNetworkTransparencyBO.java
//
//  Version:    v.1.0
//  Date:       07.02.2001
//  Status:     draft       []  10.05.2000 / Markus Moilanen (MMO)
//              proposal    []  11.01.2001 / Markus Moilanen (MMO)
//              accepted    [X] 07.02.2001 / Supervisor of the thesis
//
//  Version history:
//
//      Version     Date / Author         Comment
//      -------------------------------------------------------------------
//      0.0         10.05.2000 / MMO      File created
//      0.1         17.05.2000 / MMO      First prototype
//      0.2         19.01.2001 / MMO      Final implementation
//      0.3         03.02.2001 / MMO      The code is finished
//
//  Copyright © 2001, VTT Electronics and the author
//==========================================================================
import fi.vtt.ele.management_framework.implementation.management_layer.*;
import fi.vtt.ele.management_framework.implementation.utility.*;
import
fi.vtt.ele.management_framework.implementation.distribution_transparency.*
;
import java.net.DatagramSocket;
import java.net.SocketException;
import java.net.DatagramPacket;
import java.net.InetAddress;
import java.net.Socket;
import java.net.ServerSocket;
import java.util.Vector;
import java.io.*;
import java.util.StringTokenizer;
/**
 *<TABLE BORDER CELLPADDING=4 CELLSPACING=2 WIDTH=75%>
 *            <TR><TH ALIGN=LEFT  WIDTH=10%>Description</TH>
 *            <TD>
 * This class realises the operation defined by the NetworkTransparency
 * interface. It offers all the services which are usually required from
 * the layer under the application layer in the networked system. This
 * implementation is for the TCPIP-network, but certainly there are others
 * network type to be supported by different implementations in the
 * future, examples include DDE and the layer under the application layer
 * in the WAP or MExE protocol stack. Conveniently, the computational
 * objects always see the network as the same, no modifications are
 * required in the software when used in a different environment.
 *            </TD>
 *            <TR><TH ALIGN=LEFT>Version</TH>
 *      <TD>v1.0</TD>
 *            <TR><TH ALIGN=LEFT>Date</TH>
 *      <TD>07.02.2001</TD>
 *            <TR><TH ALIGN=LEFT>Source</TH>
 *      <TD><A href="TCPIPNetworkTransparencyBOt.java">
 * TCPIPNetworkTransparencyBO.java</A></TD>
 *            <TR><TH ALIGN=LEFT>Author</TH>
 *      <TD>Markus Moilanen (MMO)</TD>
 *</TABLE>
 * @version                v.1.0 07.02.2001
 * @author Markus Moilanen, <A HREF="http://www.iki.fi/~mmoi">
 * www.iki.fi/~mmoi</A>
 */
public class TCPIPNetworkTransparencyBO implements NetworkTransparency
{
  public NameCache name_cache = null;
  /**
    This constructor installs the network transparency class and
    initilises the name cache file as well.
  */
  public TCPIPNetworkTransparencyBO() throws Exception
  {
    name_cache = new NameCache();
  }
  /**
    The method senConnectionless sends the connectionless message in
    using the given DatagramChannel object.
    @param _with_ is a Message object to be sent.
    @param _by_ is a DatagramChannel object to be used for sending
    the data.
  */
  public void sendConnectionless(Message _with_, DatagramChannel _by_)
                                                      throws Exception
  {
    _by_.sendDatagram(getDatagram(_with_));
  }
  /**
    The method receiveConnectionless receives a message which may arrive
    in using the given DatagramChannel.
    @param _by_ is the DatagramChannel object to be used for receiving
    data.
    @param expected_size_ is the size of the expected size of the data
    of the coming message.
    @return the received data formatted as a Message object.
  */
  public Message receiveConnectionless(DatagramChannel _by_, int
expected_size_) throws Exception
  {
    Datagram d = _by_.receiveDatagram(expected_size_);
    // Maintain the name cache
    Message m = getMessage(d);
    String addr_str = setAddress(m.getFromIdentity(), d.getFromAddress());
    return m;
  }
  /**
    The method getMessage formats the contents of the given Datagram to
    the Message object.
    @param _from_ is the Datagram from which the Message is to be formed.
    @return a Message object.
  */
  public Message getMessage(Datagram _from_) throws Exception
  {
    return new MessageImplementation(_from_.getContents());
  }
  /**
    The method loadNameCache loads the name cache file from the given
    path.
    @param name_cache_file_name_ is the path extended with the file name
    expressed in the format of the underlying operating system from
    which the cache is to be loaded.
  */
  public void loadNameCache(String name_cache_file_name_) throws Exception
  {
    name_cache.loadList(name_cache_file_name_);
  }
  /**
    The method saveNameCache saves the current name cache of this network
    transparency object. The file can then be restored by the method
    loadNameCache. The information in the cache varies during the run of
    the system.
    @param name_cache_file_name_ is the path and the file name expressed
    in the format of the underlying operating system to which the cache
    is to be saved.
  */
  public void saveNameCache(String name_cache_file_name_) throws Exception
  {
    name_cache.saveList(name_cache_file_name_);
  }
  /** The method listNameCache list the current cache entries of the
      name cache.*/
  public void listNameCache() throws Exception
  {
    name_cache.writeList();
  }
  /**
    The method getAddress returns the physical address of the
    given Identity currently held by the name cache. It may happen
    that the Address cannot be resolved; therefore a new name cache
    entry for the Identity must be set. This method can consult available
    name or directory services to reach the information for maintaining
    the address information. However, in the demo,
    this approach is not currently supported for practical reasons.
    @param _of_ is the Identity of whoose address is to be resolved.
    @return the Address object.
  */
  public Address getAddress(Identity _of_) throws Exception
  {
    Address addr = null;
    InetAddress iadr = null;
    try
    {
      addr = name_cache.getEntryByIdentity(_of_).getAddress();
      iadr = InetAddress.getByName(addr.getHost());
    } catch (Exception e)
    {
      throw new Exception("NetworkTransparency: -Address for the
                                              identity'" +_of_.to_String()
      +"' is not found or cannot be interpreted. Please, configure
                                                    the cache properly.");
    }
    return addr;
  }
  /**
    The method setAddress sets the given Address for the given Identity
    in the name cache. If there is the cache entry for the given identity
    existing already, the entry will be updated according to the new
    entry.
    @param _for_ is the Address object to be associated with a new
    cache entry.
    @param _on_ is the Identity object to be associated with a new
    cache entry.
    @return a String object containíng the new cache entry formatted as
    a string.
  */
  public String setAddress(Identity _for_, Address _on_) throws Exception
  {
     if (_on_.getHost() == null)_on_.setHost
```

```
                          (InetAddress.getLocalHost().getHostAddress());
     NameCacheEntry e =  new NameCacheEntry(_for_, _on_);
     name_cache.updateEntry(e);
     return e.to_String();
  }
  /**
    The method saveNameCacheEntry can create a new name cache entry
    by means of the given text string. If a cache entry for the given
    identity already exists, the entry will be updated by the new
    information.
    @param name_cache_entry_str_ is the String object containing
    information about the identity and the address in the following
    format: "identity@host:door".
    @return an Identity object holding the new and cached entry with the
    given information.
  */
  public Identity setNameCacheEntry(String name_cache_entry_str_) throws
Exception
  {
    NameCacheEntry entry = new NameCacheEntry(name_cache_entry_str_);
    name_cache.updateEntry(entry);
    return entry.getIdentity();
  }
  /**
    The method getNameCacheEntry returns the cache entry associated with
    the given Identity.
    @param _of_ is an  Identity object to be located from the cache.
    @return the associated name cache entry.
  */
  public NameCacheEntry getNameCacheEntry(Identity _of_) throws Exception
  {
    NameCacheEntry entry = name_cache.getEntryByIdentity(_of_);
    return entry;
  }
  /**
    The method getDatagramChannel returns a new DatagramChannel object
    assigned for the given Identity. If the Identity does not have the
    proper address already, a new address will be created in using
    available information from the underlying system.
    @param identity_ is an Identity object for which a new
    DatagramChannel object
    is to be reserved.  Here we can maintain a priority based
    reservation in the future versions.
    @return a new DatagramChannel object.
  */
  public DatagramChannel getDatagramChannel(Identity identity_) throws
Exception
  {
    NameCacheEntry e = null;
    try
    {
      e = getNameCacheEntry(identity_);
    } catch (Exception r)
    {
      // there was not a name cache entry for this identity, so,
      // we will make one
      e = new NameCacheEntry(identity_.to_String() +"@"
                      +InetAddress.getLocalHost().getHostName()+":0");
      name_cache.updateEntry(e);
    }
    int port_ = e.getAddress().getDoor();
    if (port_ == 0)
    {
      DatagramChannel ch = new DatagramChannelImplementation();
      e.getAddress().setDoor(ch.getDoor());
      return ch;
    } else return new DatagramChannelImplementation(port_);
  }
  /**
    The method getDatagram returns the DatagramObject which contains
    the information from the given Message object. The new Datagram
    object is ready to be sent in using the connectionless channel,
    because it contains all the information to deliver the Datagram to
    the right Identity.
    @param _for_ is a Message object by which the Datagram
    is to be formed.
    @return a new Datagram object ready to be sent.
  */
  public Datagram getDatagram(Message _for_) throws Exception
  {
    Address from_address = getAddress(_for_.getFromIdentity());
    Address to_address = getAddress(_for_.getToIdentity());
    return new DatagramImplementation
                       (from_address, to_address, _for_.to_String());
  }
  /**
    The second method getDatagram does almost the same as the previous
    one, but only providing a Datagram object for receiving the data from
    the channel.
    @param expected_size is the expected size of the data to be received.
    @return a new Datagram object ready to store the coming data from the
    DatagramChannel.
  */
  public Datagram getDatagram(int expected_size_) throws Exception
  {
     return new DatagramImplementation(expected_size_);
  }
  /**
    The method getChannel returns the Channel object connected to the
    identity locating somewhere in the disributed environment - we do not
    even know the location before we will look for it from the name cache
    - just for curiosity.
    @param _to_ is the Identity of the other end of the transimission
    system, and to which the Channel is to be connected.
    @return a new Channel object ready to transfer data between
    two identities
    in using datastreams.
  */
  public Channel getChannel(Identity _to_) throws Exception
  {
    Address to_address = getAddress(_to_);
    return new ChannelImplementation(to_address);
  }
  /**
    The method getServerChannel reserves a some kind of "loading door"
    for the given Identity to deliver the data. The address must be
    informed about to the identity who is supposed to receive the data.
    The identity can be
    checked in the future versions.
    @param _on_ is the identity who reserves the door.
    @param door_ is the door to be reserved.  If the door_ is 0,
    a free door is to be pointed.
    @return a new ServerChannel object to be used for the data transfer
    by a Channel object in using datastreams.
  */
  public ServerChannel getServerChannel(Identity _on_, int door_) throws
Exception
  {
    ServerChannel server_channel = new ServerChannelImplementation(door_);
    // Maintain name cache
    setAddress(_on_,server_channel.getLocalAddress());
    return server_channel;
  }
  /** See the Sun's Java documentation */
  public ObjectOutputStream getObjectOutputStream(Channel _on_) throws
Exception
  {
    return new ObjectOutputStream(_on_.getOutputStream());
  }
  /** See the Sun's Java documentation */
  public ObjectInputStream getObjectInputStream(Channel _on_) throws
Exception
  {
    return new ObjectInputStream(_on_.getInputStream());
  }
}
/*************************************************************************
  Name Cache
*************************************************************************/
public class NameCache
{
  public String name_cache_file = null;
  private String default_name_cache_file = "namecache.txt";
  private Vector name_cache_list = null;
  public NameCache()
  {
    name_cache_file = default_name_cache_file;
    name_cache_list = new Vector(1,1);
  }
  public NameCache(String name_cache_file_)
  {
    name_cache_file = name_cache_file_;
    if (name_cache_file == null) name_cache_file =
                                  default_name_cache_file;
    name_cache_list = new Vector(1,1);
  }
  /** @return name_cache_list count */
  public int getCount() { return name_cache_list.size(); }
  /**
    @param v_ is a NameCacheEntry-object to add to Vector
  */
  public void addEntry(NameCacheEntry v_) throws Exception
  {
    name_cache_list.addElement(v_);
  }
  /**
    @param v_ is a NameCacheEntry-object to add to Vector
  */
  public void updateEntry(NameCacheEntry v_) throws Exception
  {
    int i = searchIndexByIdentity(v_.getIdentity());
    if (i != -1)
    {
      name_cache_list.setElementAt(v_, i);
    }  else name_cache_list.addElement(v_);
  }
  /**
    @param count_ is a index of a NameCacheEntry-object in Vector
    @return
  */
  public NameCacheEntry getEntryByCount(int count_) throws Exception
  {
    return (NameCacheEntry)name_cache_list.elementAt(count_);
  }
  public NameCacheEntry getEntryByIdentity(Identity identity_) throws
Exception
  {
    int i = searchIndexByIdentity(identity_);
    if (i != -1)
    {
      return (NameCacheEntry)name_cache_list.elementAt(i);
    }
    return null;
  }
  public boolean deleteEntryByIdentity(Identity identity_) throws
Exception
  {
    int i = searchIndexByIdentity(identity_);
    if (i != -1)
    {
      name_cache_list.removeElementAt(i);
      return true;
    }
    return false;
  }
  private int searchIndexByIdentity(Identity identity_) throws Exception
  {
    int size=getCount();
    int i = 0;
    int paluu = -1;
    if (size == 0) return paluu;
    while (i < size)
    {
      NameCacheEntry e = (NameCacheEntry)name_cache_list.elementAt(i);
      String v =  e.getIdentity().to_String();
      if (v.startsWith(identity_.to_String()))
      {
        paluu = i;
        return paluu;
      }
      i++;
    }
    return paluu;
  }
  public void writeList() throws Exception
  {
    String line = null;
    int i = 0;
    while(i < getCount())
    {
      NameCacheEntry e = (NameCacheEntry)name_cache_list.elementAt(i);
      System.out.println(e.to_String());
      i++;
    }
  }
  public void saveList(String name_cache_file_name_) throws Exception
  {
    String line = null;
    int i = 0;
    File file_ = new File(name_cache_file_name_);
    BufferedWriter bw = new BufferedWriter(new FileWriter(file_));
    while(i < getCount())
    {
      NameCacheEntry e = (NameCacheEntry)name_cache_list.elementAt(i);
      bw.write(e.to_String());
      bw.newLine();
      i++;
    }
    bw.flush();
    bw.close();
  }
  public void loadList(String name_cache_file_name_) throws Exception
  {
    String line = null;
    File file_ = new File(name_cache_file_name_);
    if (file_.canRead())
    {
      BufferedReader br = new BufferedReader(new FileReader(file_));
      while ((line = br.readLine()) != null)
      {
        NameCacheEntry e = new NameCacheEntry(line);
        name_cache_list.addElement(e);
      }
      br.close();
    } else
```

```
    {
      throw new Exception("NetworkTransparency: -The name cache file
'"+name_cache_file_name_+"' cannot be opened.");
    }
  }
}
/**************************************************************************
  Name Cache Entry
**************************************************************************/
public class NameCacheEntry
{
  private Address address = null;
  private Identity identity = null;
  public String entry_string = "unknown@localhost:0";
  public NameCacheEntry(){};
  public NameCacheEntry(Identity identity_, Address address_) throws
Exception
  {
    identity = identity_;
    address = address_;
  }
  public NameCacheEntry(String entry_str_) throws Exception
  {  from_String(entry_str_);   }
  public void from_String(String row_) throws Exception
  {
    Object view_parameters = new StringTokenizer( row_, "@" );
    String identity_str = ((StringTokenizer) view_parameters).nextToken();
    String address_str = ((StringTokenizer) view_parameters).nextToken();
    identity = new IdentityImplementation(identity_str);
    address = new AddressImplementation(address_str);
  }
  public Identity getIdentity() {return identity;}
  public Address getAddress() {return address;}
  /**  @return String including fields as a text.  */
  public String to_String() throws Exception {return
(identity.to_String()+"@"+address.to_String()); }
}
```

Utility package

```
//==========================================================================
//  VTT Electronics                             Java Class Definition File
//
//  Project:    DIT_MMO
//  Task:       Management Framework of Distributed Software Objects
//              and Components.
//  Package:
//  File:       Multible class definition files of the implementations of
//              the utility package: AddressImplementation.java,
//              ChannelImplementation.java, DatagramImplementation.java,
//              DatagramChannelImplementation.java,
//              DecisionImplementation.java,
//              IdentityImplementation.java,
//              ManagementActionImplementation.java,
//              MessageImplementation.java, ReturnCodeImplementation.java,
//              ServerChannelImplementation.java, and DiskFile.java.
//
//  Version:    v.1.0
//  Date:       07.02.2001
//  Status:     draft       []  10.05.2000 / Markus Moilanen (MMO)
//              proposal    []  11.01.2001 / Markus Moilanen (MMO)
//              accepted    [X] 07.02.2001 / Supervisor of the thesis
//
//  Version history:
//      Version     Date / Author         Comment
//      ---------------------------------------------------------------
//      0.0         10.05.2000 / MMO      File created
//      0.1         17.05.2000 / MMO      First prototype
//      0.2         19.01.2001 / MMO      Final implementation
//      0.3         03.02.2001 / MMO      The code is finished
//
//  Copyright © 2001, VTT Electronics and the author
//
//=====================================================================
import fi.vtt.ele.management_framework.implementation.management_layer.*;
import fi.vtt.ele.management_framework.implementation.utility.*;
import
fi.vtt.ele.management_framework.implementation.distribution_transparency.*
;
import java.net.DatagramSocket;
import java.net.SocketException;
import java.net.DatagramPacket;
import java.net.InetAddress;
import java.net.Socket;
import java.net.ServerSocket;
import java.util.Vector;
import java.io.*;
import java.util.StringTokenizer;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.File;
/*********************************************************************
  Address implementation
*********************************************************************/
public class AddressImplementation implements Address
{
  private String host_str = null;
  private int door = 0;
  private String address_str = null;
  public AddressImplementation(String host_str_, int door_) throws
Exception
  {
    host_str = host_str_;
    door = door_;
    address_str = toString();
  }
  public AddressImplementation(String address_) throws Exception
  {
    from_String(address_);
  }
  public AddressImplementation()  throws Exception {}
  public void setHost(String host_str_) { host_str = host_str_; }
  public String getHost() { return host_str;}
  public void setDoor(int door_) { door = door_;}
  public int getDoor() {return door;}
  /** Wrap the information of this address to send by  string */
  public String to_String() { return     new String(host_str+':'+door); }
  /**  Splits a string which was in the given string.
    @return true, if operation was succesful, else false.
  */
  private void from_String(String s) throws Exception
  {
    boolean ok = true;
    host_str = s.substring(0,s.indexOf(":"));
    String p = s.substring(s.indexOf(":")+1,s.length());
    try
    {
      Integer dummy = new Integer(p);
      door = dummy.intValue();
    } catch ( NumberFormatException e) {ok = false;}
    address_str = host_str+':'+door;
    if (!ok) throw new Exception();
  }
}
/**************************************************************************
  Channel implementation
**************************************************************************/
public class ChannelImplementation implements Channel
{
  Socket socket = null;
  public ChannelImplementation(Address to_address_) throws Exception
  {
    socket = new Socket(InetAddress.getByName(to_address_.getHost()),
to_address_.getDoor(), true);
  }
  public ChannelImplementation(Socket socket_) throws Exception
  {
    socket = socket_;
  }
  public InputStream getInputStream() throws Exception
  {
    return socket.getInputStream();
  }
  public OutputStream getOutputStream() throws Exception
  {
    return  socket.getOutputStream();
  }
  public void close()
  {
    try
    {
      socket.close();
    } catch (Exception e) {}
  }
}
/**************************************************************************
  Datagram implementation
**************************************************************************/
public class DatagramImplementation implements Datagram
{
  private int default_receive_buffer_size = 512;
  private DatagramPacket packet = null;
  private byte[] buf;
  private Address from_address = null;
  private Address to_address = null;
  private String contents = null;
  /**
    Constructs a Datagram for receiving packets.
    @param size_ is the data buffer size for receive data.
  */
  public DatagramImplementation(int size_) throws Exception
  {
    if (size_ <= 0) size_ = default_receive_buffer_size;
    buf = new byte[size_];
    packet = new DatagramPacket(buf, buf.length);
  }
  /**
    Constructs a Datagram for sending datagram to the specified address
    with specified contents.
    @param from_address_ is the Address of the sender.
    @param to_address_ is the Address of the receiver.
    @param contents_ is the contents of the datagram.
  */
  public DatagramImplementation(Address from_address_, Address
to_address_, String contents_) throws Exception
  {
    from_address = from_address_;
    to_address = to_address_;
    contents = contents_;
    String s =
"<"+from_address.to_String()+"><"+to_address.to_String()+">"+contents;
    buf = new byte[s.length()];
    s.getBytes(0,s.length(),buf,0);
    packet = new DatagramPacket(buf, buf.length,
InetAddress.getByName(to_address_.getHost()),to_address_.getDoor());
  }
  public DatagramImplementation(DatagramPacket packet_) throws Exception
  {
    packet = packet_;
    deCodePacket();
  }
  public void deCodePacket() throws Exception
  {
    String s = new String(packet.getData(),0);
    String from_address_str_ = s.substring(1,s.indexOf(">"));
    from_address = new AddressImplementation(from_address_str_);
    String t = s.substring(s.indexOf(">")+1,s.length());
    String to_address_str_ = t.substring(1,t.indexOf(">"));
    to_address = new AddressImplementation(to_address_str_);
    String c = t.substring(t.indexOf(">")+1,t.length());
    contents = c.trim();
  }
  public Address getFromAddress() {return from_address;}
  public Address getToAddress() {return to_address;}
  public String getContents() {return contents;}
  public void setToAddress(Address to_address_) {to_address =
(AddressImplementation)to_address_;}
  public void setFromAddress(Address from_address_) {from_address =
(AddressImplementation)from_address_;}
  public void setContents(String contents_) {contents = contents_;}
  public DatagramPacket getDatagramPacket() {return packet;}
  public String to_String(){return
"<"+from_address.to_String()+"><"+to_address.to_String()+">"+contents;}
}
/*************************************************************************
  DatagramChannel implementation
*************************************************************************/
public class DatagramChannelImplementation implements DatagramChannel
{
  private int port = 0;
  private DatagramSocket datagramsocket = null;
  public DatagramChannelImplementation() throws SocketException
  {
    datagramsocket = new DatagramSocket();
    port = datagramsocket.getLocalPort();
  }
  public DatagramChannelImplementation(int port_) throws SocketException
  {
    port = port_;
    datagramsocket = new DatagramSocket(port_);
  }
  public void sendDatagram(Datagram datagram_) throws Exception
  {
    DatagramImplementation p = (DatagramImplementation)datagram_;
    datagramsocket.send(p.getDatagramPacket());
  }
  public Datagram receiveDatagram(int expected_size_) throws Exception
  {
    DatagramImplementation d = new DatagramImplementation(expected_size_);
    datagramsocket.receive(d.getDatagramPacket());
    d.deCodePacket();
    return (Datagram)d;
  }
  public void setTimeout(int timeout_) throws Exception
  {
    datagramsocket.setSoTimeout(timeout_);
  }
  public int getDoor() {return port;}
}
/*************************************************************************
  Decision Implementation
*************************************************************************/
public class DecisionImplementation implements Decision
```

```
{
  public boolean DEMANDS = false;
  public boolean SUPPORTS = false;
  public boolean JUDGMENT = false;
  public DecisionImplementation(boolean demands_, boolean supports_,
boolean judgment_) throws Exception
  {
    DEMANDS = demands_;
    SUPPORTS = supports_;
    JUDGMENT = judgment_;
  }
  public boolean DEMANDS() { return DEMANDS; }
  public boolean SUPPORTS() { return SUPPORTS; }
  public boolean JUDGMENT() { return JUDGMENT; }
}
/*************************************************************************
  Identity implementation
*************************************************************************/
public class IdentityImplementation implements Identity
{
  public String identity_str = null;
             public IdentityImplementation(String identity_str_) throws
Exception
  {
    identity_str = identity_str_;
  }
  public IdentityImplementation(Identity identity_) throws Exception
  {
    identity_str = new String(identity_.to_String());
  }
  public void from_String(String identity_str_)
  {
    identity_str = identity_str_;
  }
  public String to_String() {return identity_str;}
}
/*************************************************************************
  Information Branch is implemented in the HelloWorldApplication class.
**************************************************************************
  Information Root is not implemented in the demo.
**************************************************************************
  ManagementAction implementation
*************************************************************************/
public class ManagementActionImplementation extends MessageImplementation
implements ManagementAction
{
  protected int intvalue = 0;
  protected long longvalue = 0;
  protected Object object = null;
  public ManagementActionImplementation() throws Exception {}
  public ManagementActionImplementation(int intvalue_, long longvalue_,
Object object_, Message message_) throws Exception
  {
    intvalue = intvalue_;
    longvalue = longvalue_;
    object = object_;
    from_identity = message_.getFromIdentity();
    to_identity = message_.getToIdentity();
    prefix = message_.getPrefix();
    body = message_.getBody();
    code = message_.getCode();
  }
  public ManagementActionImplementation(String management_action_str_)
throws Exception
  {
    if (!from_String(management_action_str_)) throw new
Exception("Parameter cannot be interpreted.");
  }
  public void setIntvalue(int intvalue_) { intvalue = intvalue_; }
  public void setLongvalue(long longvalue_) { longvalue = longvalue_; }
  public void setObject(Object object_) { object = object_; }
  public boolean adaptMessage(Message message_)
  {
    try
    {
      from_identity = new
IdentityImplementation(message_.getFromIdentity());
      to_identity = new IdentityImplementation(message_.getToIdentity());
      adaptReturnCode(message_.getPrefix(), message_.getBody(),
message_.getCode());
      return true;
    } catch (Exception e) { return false; }
  }
  public int getIntvalue() {return intvalue;}
  public long getLongvalue() {return longvalue;}
  public Object getObject() {return object;}
  public Message createMessage() {return (Message)super;}
  /**  The next method will be corrected in the next version. */
  public String to_String() {return
/*intvalue+"_"+longvalue_"+object.toString()+"_"+*/super.to_String();}
  public boolean from_String(String management_action_str_)
  {
    try
    {  /* will be developed */
      super.from_String(management_action_str_);
    } catch (Exception e) {return false;}
    return true;
  }
}
/**************************************************************************
  Message implementation
**************************************************************************/
public class MessageImplementation extends ReturnCodeImplementation
implements Message
{
  protected Identity from_identity = null;
  protected Identity to_identity = null;
  public MessageImplementation() throws Exception {}
  public MessageImplementation(Identity from_identity_, Identity
to_identity_,   ReturnCode returncode_) throws Exception
```

```
  {
    from_identity = from_identity_;
    to_identity = to_identity_;
    prefix = returncode_.getPrefix();
    body = returncode_.getBody();
    code = returncode_.getCode();
  }
  public MessageImplementation(Identity from_identity_, Identity
to_identity_, String prefix_, String body_, String code_) throws Exception
  {
    from_identity = from_identity_;
    to_identity = to_identity_;
    prefix = prefix_;
    body = body;
    code = code_;
  }
  public MessageImplementation(String message_str_) throws Exception
  {
    if (!from_String(message_str_)) throw new Exception("Parameter cannot
be interpreted.");
  }
  public boolean adaptReturnCode(String prefix_, String body_, String
code_)
  {
    try
    {
      prefix = new String(prefix_);
      body = new String(body_);
      code = new String(code_);
      return true;
    } catch (Exception e) {  return false; }
  }
  public boolean adaptReturnCode(ReturnCode returncode_)
  {
    try
    {
      prefix = new String(returncode_.getPrefix());
      body = new String(returncode_.getBody());
      code = new String(returncode_.getCode());
      return true;
    } catch (Exception e) { return false; }
  }
  public ReturnCode createReturnCode()
  {
    try
    {
      return new ReturnCodeImplementation(prefix, body, code);
    } catch (Exception e) {return null;}
  }
  public Identity getFromIdentity() {return (Identity)from_identity;}
  public Identity getToIdentity() {return (Identity)to_identity;}
  public void setToIdentity(Identity to_identity_) {to_identity =
(IdentityImplementation)to_identity_;}
  public void setFromIdentity(Identity from_identity_) {from_identity =
(IdentityImplementation)from_identity_;}
```

```
  public String to_String() { return
"<"+from_identity.to_String()+"><"+to_identity.to_String()+">"+super.to_St
ring();}
  public boolean from_String(String message_str_)
  {
    try
    {
      String from_identity_str_ =
                   message_str_.substring(1,message_str_.indexOf(">"));
      from_identity = new IdentityImplementation(from_identity_str_);
      String t = message_str_.substring
                      (message_str_.indexOf(">")+1,message_str_.length());
      String to_identity_str_ = t.substring(1,t.indexOf(">"));
      to_identity = new IdentityImplementation(to_identity_str_);
      String c = t.substring(t.indexOf(">")+1,t.length());
      super.from_String(c);
    } catch (Exception e) {return false;}
    return true;
  }
  public boolean swapIdentities()
  {
    try
    {
      Identity temp = new IdentityImplementation(to_identity);
      to_identity = from_identity;
      from_identity = temp;
      return true;
    } catch (Exception e) {return false;}
  }
}
/**************************************************************************
  ReturnCode implementation
**************************************************************************/
public class ReturnCodeImplementation implements ReturnCode
{
  protected String prefix = null;
  protected String body = null;
  protected String code = null;
  protected String fix = "+";
  public ReturnCodeImplementation() throws Exception
  {
    setDENIED();
    body = "DEFAULT";
    setFAILED();
  }
  public ReturnCodeImplementation(ReturnCode returncode_) throws Exception
  {
    prefix = returncode_.getPrefix();
    body = returncode_.getBody();
    code = returncode_.getCode();
  }
  public ReturnCodeImplementation(String prefix_, String body_, String
code_) throws Exception
  {
    prefix = prefix_;
    body = body_;
    code = code_;
  }
  public ReturnCodeImplementation(String returncode_str_) throws Exception
  {
    if (!from_String(returncode_str_)) throw new Exception("Parameter
cannot be interpreted.");
  }
  public boolean isQUERY()
  {
    if (prefix.compareTo(query)==0) return true; else return false;
  }
  public boolean isMAXIMUM()
  {
    if (prefix.compareTo(maximum)==0) return true; else return false;
  }
  public boolean isBUSY()
  {
    if (prefix.compareTo(busy)==0) return true; else return false;
  }
  public boolean isGRANTED()
  {
    if (prefix.compareTo(granted)==0) return true; else return false;
  }
  public boolean isDENIED()
  {
    if (prefix.compareTo(denied)==0) return true; else return false;
  }
  public void setQUERY() { prefix = query; }
  public void setMAXIMUM() { prefix = maximum; }
  public void setBUSY() { prefix = busy; }
  public void setGRANTED() { prefix = granted; }
  public void setDENIED() { prefix = denied; }
  public boolean isREPRESENTATIVE_PROXY()
  {
    if (body.compareTo(representative_proxy)==0) return true; else return
false;
  }
  public void setREPRESENTATIVE_PROXY() {body = representative_proxy;}

  public boolean isCLASS_BYTE_CODE_RESOURCE()
  {
    if (body.compareTo(class_byte_code_resource)==0) return true; else
return false;
  }
  public void setCLASS_BYTE_CODE_RESOURCE() {body =
class_byte_code_resource;}
  /** @return boolean true if the code_ is SUCCESS-code, else returns
false. */
  public boolean isSUCCESS()
  {
    if (code.compareTo(success)==0) return true; else return false;
  }
  public boolean isFAILED()
  {
    if (code.compareTo(failed)==0) return true; else return false;
  }
  public boolean isINFOCODE()
  {
    if (code.compareTo(infocode)==0) return true; else return false;
  }
  public void setSUCCESS() { code = success; }
  public void setFAILED() { code = failed; }
  public void setINFOCODE() { code = infocode; }
  public String getString() { return to_String(); }
  public String getPrefix() { return prefix; }
  public String getBody() { return body; }
  public String getCode() { return code; }
  public void setPrefix(String prefix_) { prefix = prefix_;}
  public void setBody(String body_) { body = body_;}
  public void setCode(String code_) { code = code_;}
  public String to_String() { return prefix+fix+body+fix+code; }
  public boolean from_String(String returncode_str_)
  {
    try
    {
      StringTokenizer q = new StringTokenizer(returncode_str_,fix);
      prefix=q.nextToken();
      body=q.nextToken();
      code=q.nextToken();
      return true;
    } catch (Exception e) {return false;}
  }
}
/*************************************************************************
  ServerChannel implementation
*************************************************************************/
public class ServerChannelImplementation implements ServerChannel
{
  private ServerSocket server_socket = null;
  private int door = 0;
  private Socket socket = null;
  public ServerChannelImplementation(int door_) throws Exception
  {
    server_socket = new ServerSocket(door_);
    door = server_socket.getLocalPort();
  }
  public Channel accept() throws Exception
  {
    socket = server_socket.accept();
    return new ChannelImplementation(socket);
  }
  public int getLocalDoor() throws Exception { return door; }
  public Address getLocalAddress() throws Exception
  {
    String addr =
server_socket.getInetAddress().getLocalHost().getHostAddress();
    return new AddressImplementation(addr, door);
  }
  public void close() throws Exception
  {
    server_socket.close();
  }
}
/*************************************************************************
  DiskFile
*************************************************************************/
public class DiskFile implements Serializable
{
  private byte[] databuffer = null;
  public String filename = null;
  public DiskFile() { }
  public DiskFile(String filename_) throws Exception
  {
    if (!fromDisk(filename_)) throw new Exception("File not found.");
  }
  public boolean fromDisk(String filename_)
  { filename = filename_;
    try
    {
      File readfile = new File(filename);
      FileInputStream file_reader = new FileInputStream(readfile);
      databuffer = new byte[(int)readfile.length()];
      int readbytes = file_reader.read(databuffer);
      file_reader.close(); return true;
    } catch (IOException e)
    {
      System.out.println("DiskFile, fromDisk: I/O error."); return false;
    }
  }
  public boolean toDisk()
  {
    try
    { FileOutputStream file_writer = new FileOutputStream(filename);
      file_writer.write(databuffer);
      file_writer.close();
      file_writer.flush();
      return true;
    } catch (IOException e)
    {
      System.out.println("DiskFile, toDisk: I/O error."); return false;
    }
  }
}
```

End of the document.

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 442
VTT–PUBS–442

Author
Moilanen, Markus

Title

Management framework of distributed software objects and components

Abstract

The deployment of software applications without making any arrangements beforehand in the target platform sets strict requirements for the management of the applications and the management of the platform in which the application will be executed. In this work, "objective quality thinking" concerning the work process has been chosen to gain a solution for this problematic issue. In the work, the management framework of the distributed applications is developed. Application software is understood to be any useful piece of software that can be provided.

The developed management framework observes different dimensions of the management. Dimension is such a feature group of the management which can be implemented independently of other feature groups. These dimensions include for example the advertisement of a component, distribution of a component, resource management of a component, and security of a component. The dimensions are presented starting from the conceptual level ending to the explicit design models. These design models can be implemented for any application-specific purpose by implementing the presented application programming interfaces of the models. Lastly in the work, an example of how to apply the management to an application is presented. The case considers an application which distributes itself according to its associated distribution policy, and which is implemented for the Internet environment in the Java programming language.

The presented solution of the management can be applied to any data objects, including passive objects. If extended to its full design, the solution would be a starting point for a whole new software management technology.

Keywords
software processes, software design, quality methods, requirement specifications, conceptual planning, application layer, management layer, component architecture, UML

Activity unit
VTT Electronics, Embedded Software, Kaitoväylä 1, P.O.Box 1100, FIN–90571 OULU, Finland

ISBN 951–38–5863–4 (soft back ed.) 951–38– 5864–2 (URL:http://www.inf.vtt.fi/pdf/)	Project number

Date	Language	Pages	Price
September 2001	English	153 p. + 45 app.	C

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 ELEKTRONIIKKA – VTT ELEKTRONIK – VTT ELECTRONICS

VTT PUBLICATIONS

293 Karioja, Pentti. Integrated optics for instrumentation applications. 1996. 44 p. + app. 65 p.

298 Ailisto, Heikki. CAD model-based planning and vision guidance for optical 3D co-ordinate measurement. 1997. 70 p. + app. 63 p.

308 Huuskonen, Pertti J. Model-based explanation of plant knowledge. 1997. 173 p. + app. 59 p.

313 Heikkinen, Marko. A concurrent engineering process for embedded systems development. 1997. 93 p.

318 Latvakoski, Juhani. Integration test automation of embedded communication software. 1997. 98 p. + app. 28 p.

344 Soininen, Juha-Pekka, Huttunen, Tuomo, Saarikettu, Janne, Melakari, Klaus, Ong, Andy & Tiensyrjä, Kari. InCo – An interactive codesign framework for real-time embedded systems. 1998. 206 p.

346 Seppänen, Veikko, Alajoutsijärvi, Kimmo & Kurki, Matti. Competence-based evolution of contractual R&D relationships. 1998. 70 p.

347 Anttila, Tommi. A haptic rendering system for virtual handheld electronic products. 1998. 69 p.

355 Melakari, Klaus. A VHDL simulator in a co-verification environment. 1998. 91 p. + app. 6 p.

365 Taramaa, Jorma. Practical development of software configuration management for embedded systems. 1998. 147 p. + app. 110 p.

370 Laitinen, Jyrki. Evaluation of imaging in automated visual web inspection. 1998. 93 p. + app. 86 p.

373 Rantala, Juha. Sol-gel materials for photonic applications. 1998. 50 p. + app. 48 p.

375 Niemelä, Eila, Korpipää, Tomi & Tuominen, Arno. Embedded middleware: State of the art. 1999. 102 p. + app. 7 p.

382 Rauma, Tapio. Fuzzy modeling for industrial systems. 1999. 97 p. + app. 40 p.

392 Seppänen, Veikko, Alajoutsijärvi, Kimmo & Eriksson, Päivi. Projects or products: seeking for the business logic of contract R&D. 1999. 110 p. + app. 104 p.

402 Niemelä, Eila. A component framework of a distributed control systems family. 1999. 188 p. + app. 68 p.

406 Korhonen, Jukka, Salmela, Mika & Kalaoja, Jarmo. The reuse of tests for configured software products. 2000. 67 p.

407 Rusanen, Outi. Adhesives in micromechanical sensor packaging. 74 p. + app. 54 p.

409 Rahikkala, Tua. Towards virtual software configuration management. A case study. 2000. 110 p. + app. 57 p.

416 Mäkäräinen, Minna. Software change management processes in the development of embedded software. 2000. 185 p. + app. 56 p.

423 Saloniemi, Heini. Electrodeposition of PbS, PbSe and PbTe thin films. 2000. 82 p. + app. 53 p.

426 Järvinen, Janne. Measurement based continuous assessment of software engineering processes. 2000. 97 p. + app. 90 p.

427 Dobrica, Liliana & Niemelä, Eila. A strategy for analysing product line software architectures. 2000. 124 p.

442 Moilanen, Markus. Management framework of distributed software objects and components. 2001. 153 p. + app. 45 p.