Quality-oriented exploration techniques for component-based architectures

Zhang, J.

Award date:
2007

TECHNISCHE UNIVERSITEIT EINDHOVEN
Department of Mathematics and Computer Science

MASTER'S THESIS

QUALITY-ORIENTED EXPLORATION TECHNIQUES FOR COMPONENT-BASED ARCHITECTURES

by

Jiang Zhang

Graduation Committee:

Dr. M. R.V. Chaudron
Dr. M. Voorhoeve
Ir. A. Klomp (LogicaCMG)
Dr. J.C.S.P. van der Woude

Eindhoven, August 2005

ABSTRACT

Software systems are becoming more and more important for all industrial fields. Software developers are asked to produce high-quality software products, which require lower costs and resource consumption. However, some designs are good for some quality attributes while bad for others. Therefore we need a technique to evaluate multiple quality properties in order to motivate design trade-offs.

Our project aims to develop quantitative techniques and to build a tool to assess a number of quality properties of architectural designs. This tool calculates the effect of applying the alternative architectures to relevant quality properties. We use a methodology that applies multiple analysis methods to assess different quality properties of component-based architecture, which we call DAX (**De**sign, **A**nalyze, **eX**plore, [1]). In this paper we apply the DAX methodology to two case studies, assessing the reliability, performance and cost criteria. One case study is a car navigation system, while another one is an iTV system.

TABLE OF CONTENTS

ACKNOWLEDGMENTS

In 2003 I started my master study program of computer science at the Technical University of Eindhoven (TUE). From the various courses and projects that were encountered throughout the terms I was always fond of the ones related to software architecting. My first lecture in the Netherlands was the course software architecting given by Michel Chaudron. From 2004 July, I worked in Laquso as a student assistant for a project called "UML design refactoring with metrics", supervised by Michel. After having accomplished it, we talked about a project topic about quantitative assessment methods for component-based system architecture. The contacts of Arjen Klomp led to the decision of the subject for my graduation project in LogicaCMG.

I would like to acknowledge with particular gratitude the assistance of my tutor Michel Chaudron, my advisor Arjen Klomp, who played an advisory roll in this project, and Marc Voorhoeve who agreed to be my graduation supervisor. I am grateful to Egor Bondarev, the author of the original RTIE tool for taking part in several meetings and for his technical assistance. Furthermore I would like to thank Dr. Jaap van der Woude for taking place in my graduation committee. For useful discussions I would like to thank Ad Aerts, Natalia Sidorova, Klaas Wijbrans and Arne Rippe. I would like to thank my family for supporting me through my entire study and providing me a basis for completing this project. Finally I would like to thank my girlfriend Xue for her understanding, support and great patience with me even after we haven't seen each other for two years. I never forget her efforts and sacrifice.

ABBREVIATIONS

CA	Conditional Access
CBA	Component-based Architecture
COTS	Commercial Off-The-Shelf [31]
DAX	Design, Analyze, eXplore
IEEE	Institute of Electrical and Electronics Engineer
ISO	International Standards Organization
iTV	Interactive Television
MIPS	Million Instructions Per Second
OO	Object-Oriented
RDS	Radio Digital Signal
RTIE	Robocop Integration Environment
STB	Set-Top Box
TMC	Traffic Message Channel
UML	Unified Modeling Language
XML	Extensible Markup Language

1. Introduction

The results of this thesis document were obtained during the graduation project with the topic **Techniques for exploring architectural design alternatives from extra-functional and quality perspectives.** In this chapter we will discuss the main research question of the project in the first section. In the second section, we will present related work to put this project into perspective and finally we will give an outline of the next chapters of this document.

1.1 Research Question

Software systems are becoming more and more important for all industrial fields. But with the increase of demands on and impacts of software, the complexity of systems increases greatly as well. Software developers are asked to produce high-quality software products, which require lower costs and resource consumption.

It is well known that design decisions should be made at an early stage of the software development process. The first step in the software lifecycle is to make an excellent architecture. A main purpose of an architecture model is to structure the design problem and to find solution. It also helps stakeholders to communicate about the problem and solution. For a system, normally architects can find some possible alternative designs. However, some designs are good for some quality attributes while bad for others. Software architects are then required to achieve balance between the different quality requirements (such as performance, reliability, cost etc) for a satisfactory system.

To solve the problem, the architect should construct models of architectural alternatives and assess their quality properties, which are significant to the system. Therefore we need a quantitative technique to evaluate multiple quality properties in order to motivate design trade-offs. Also we need a tool to assess a number of quality properties (reliability and cost are mainly focused in this thesis) of architectural designs. This tool calculates the effect of applying the alternative architectures to relevant quality properties. We use the so-called methodology DAX (**D**esign, **A**nalyze, e**X**plore, [1]) that uses multiple analysis methods to assess different quality properties of component-based architectures. Our analysis approach is based on the evaluation of a number of key scenarios in system architecture. One of the advantages of using scenarios is the possibility to scale the evaluation coverage of the system.

Before this project, we already had a resource model and performance-relevant exploration methodology for Robocop CBA [19]. One of the main contributions of this project is to create two new quality models for reliability and cost calculation of Robocop CBA.

The contributions of the project are the following:

A. The introduction of Introduce Use case and deployment diagrams into Robocop Component-Based Architecture (CBA) [19]. Use case diagrams are used as a high level organization of scenarios. Deployment diagrams describe the physical deployment of the software components; it allows mapping software components onto hardware nodes, thereby increasing design flexibility and efficiency, and helping architects to select better alternative architectures.

B. The creation of Reliability Models and Cost Models for Robocop CBA. These models enable reliability and cost assessments of Robocop CBA.

C. The methodology supports the design of multiprocessor systems or hot-standby computers.

D. Our reliability and cost assessment methods take the hardware factors into account; therefore they extend the old quality assessment methodology of Robocop CBA from a software-focused to overall system range.

1.2 Related Work

Multi-dimensional design space exploration is a challenging topic that has been extensively addressed in system architecture research. Main ideas based on evolutionary algorithms are presented in [2]. Specific approaches are used for different application domains. The "Spade" technique [3] provides a methodology for architecture exploration of heterogeneous signal processing systems. A framework based on real-time calculus [4] is proposed for design space exploration of network processor architectures. Related work on the simulation-based design exploration of system-on-chip communication architectures is presented in [5] and [6].

For a software component-based system domain, few techniques exist: micro-architecture modeling [7] and scenario-based static performance evaluation [8]. The above-mentioned approaches are focused mostly on the performance-cost-power trade-off. The research on the early assessment of the reliability quality attribute is rarely linked to the design space exploration

topic. One exception is the technique based on symbolic search and multi-granular simulation [9]. Scenario-based approaches for early assessment of component-based software architectures are described in [10] and [11]. A recent method based on Markov models for reliability estimation of component-based systems is given in [12]. [34] introduced a methodology of reliability prediction based on service effects state machines and Markov chains. None of the above-mentioned approaches address the multi-objective design space exploration for software component-based multiprocessor systems.

1.3 Outline of Thesis

Chapter 2 introduces the preliminary knowledge for this project. We introduce software architecture and quality attributes of software systems. Robocop CBA and the design space exploration methodology are presented. Robocop is the basis for our quality exploration research.

Chapter 3 introduces the data models for our quality prediction. We need both structural data and quality data as inputs. We explain in this chapter a car navigation case study and its models that serve as a foundation of formal reliability calculation.

Chapter 4 describes the algorithm of reliability, cost calculation, and a case study that is performed in this project. Chapter 5 presents an iTV system as case study, explains its input models and multi-objective evaluation results for multiple architecture solutions.

Chapter 6 contains the description of the tool implementation. Chapter 7 contains the conclusions of the project. Chapter 8 gives recommendations for future research.

2. Preliminaries

2.1 Software Architecture

Software architecture is a high-level abstraction of a software system, which contains information to conduct the whole lifestyle of software development. Like the building architects do the purpose, themes, materials and concept of a structure, software architects develops concepts and plans for the high-level design of a software product, e.g. deployment of components, packages and layers, communication and interface between them, high-level business object operations, logic and flows. [13]

Software architecture is not only the basis of further design and implementation, but also the common communication platform between the stakeholders, such as end-users, product manager, software engineers, etc. It is the foundation stone of building a successful software system. [14]

2.1.1 Overview of Definitions

With the rapid growth of software techniques, the definitions of software architecture also vary very quickly. Now from different perspectives, experts have given many understandings of general software architectures. Below we will present two of them [15]:

A recent definition is from Bass's book [13], "The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them". According to this definition, software architecture is a compromise between system structures and a set of behavior processes.

Booch, Rumbaugh, and Jacobson gave a definition [16] that states that: "An architecture is the set of significant decisions about the organization of a software system, the selection of the structural elements and their interfaces by which the system is composed, together with their behavior as specified in the collaborations among those elements, the composition of these structural and behavioral elements into progressively larger subsystems, and the architectural style that guides this organization---these elements and their interfaces, their collaborations, and their composition". According to this second definition, software architecture is the selection, organization and composition of the static and dynamic elements.

At the First International Workshop on Architectures for Software Systems [17], Mary Shaw provided the different views of software architecture:

- **Structural models** are composed of components, connections among those components, plus some other aspects, e.g. requirements, constraints.

- **Framework models** are similar to the structural view, but their primary emphasis is on the coherent structures of the whole system, as opposed to concentrating on its composition. Framework models often target specific domains or problem classes.

- **Dynamic models** emphasize the behavioral quality of systems. "Dynamic" may refer to changes in the overall system configuration, setting up or disabling pre-enabled communication or interaction pathways, or the dynamics involved in the progress of the computation, such as changing data values.

- **Process models** focus on construction of the architecture, and the steps or process involved in that construction. In this view, architecture is the result of following a process script.

Joined all together, they form a consensus view of software architecture.

2.1.2 Kruchten's 4+1 View Model

As we mentioned earlier, software architecture is the common communication platform for the stakeholders. Different stakeholders have various needs to the software system. On the other hand, they also have different backgrounds and different perspectives on the software architecture, i.e. architects cannot expect the end-users to review their component designs, behavior models or other technical details. The sole concern of the end-users is if the architecture fits into their functional or non-functional requirements. For the software programmers, they may not understand how the whole system works (especially for the large domain-specific information system); they would like to have clear technical specifications about the interfaces or classes' design, and flowchart of the program. It is a better way to classify the different views of architecture and design them respectively. And it is actually infeasible to present every aspects of the whole architecture design, static and dynamic, in a big diagram.

Philippe Kruchten of Rational Software proposes his "4+1 View Model" [18]. Kruchten's definition of architecture is that it is a high level structure addressing the functional and non-functional requirements of system. He proposed five models for different stakeholders from respective views.

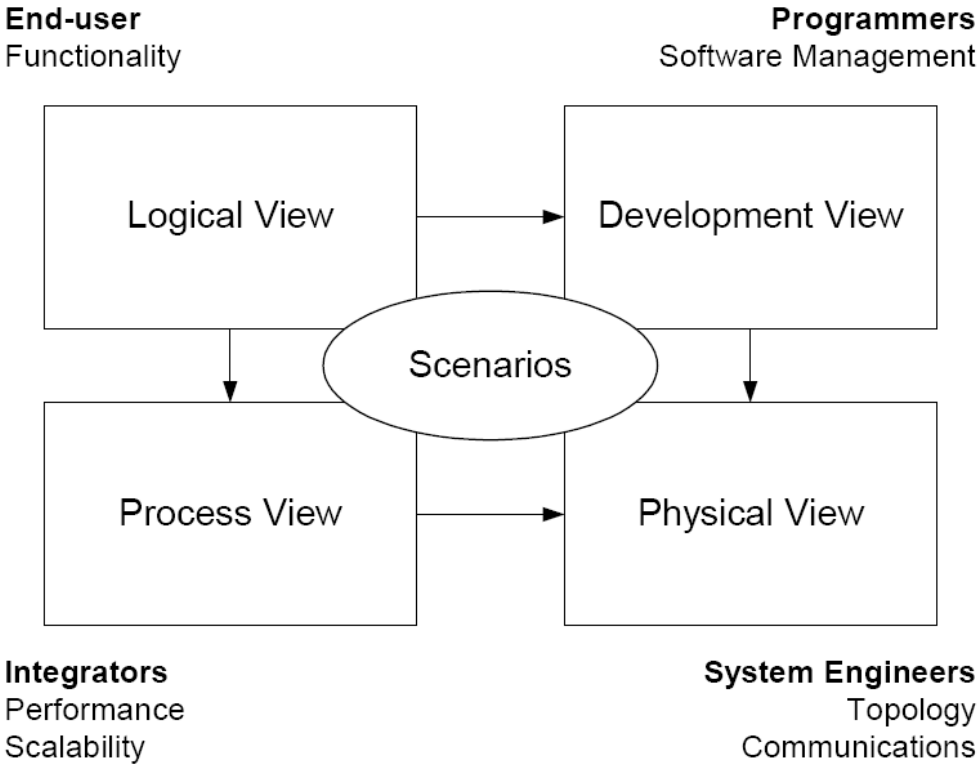


Figure 1- "4+1 View Model of Software Architecture" (source: [18])

The model represented in Figure 1 is made of five main views:

1. **The logical view**. It is the model of the decomposition of system structure. It primarily supports the functional requirements to its users and is composed of a set of key abstractions taken from the problem domain in the form of class diagram in UML. It is also used to identify common mechanisms and design elements from the various parts of the system.

2. **The Process View**. A process is a set of tasks that comes from an executable unit. A task is a separate thread of control, which individually runs on a physical node. The process view represents several levels of abstraction at which the process architecture can be tactically controlled. Each level addresses different concerns. The process architecture contains the information of non-functional requirements, and issues of concurrency and distribution.

3. **Development View**. Development view organizes the software modules in a hierarchy of layers and shows the relationships between the modules, subsystems and layers. Development architecture will be complete only if all the elements of the software have

been identified. So it also serves as the basis for requirement allocation, and for management purpose, i.e. cost evaluation, planning, etc.

4. **Physical View**. It serves as a mapping from software to physical deployment. To complete it, we need to identify the physical devices that we need to execute software, e.g. a network of computers, or processing nodes. The physical architecture is expected to be highly flexible from the point of view of the source code so that we could have different physical configurations for different customers.

5. **Scenario View**. The elements in the above four views are strongly linked by a set of scenarios. In UML we call the scenarios "use cases". They are described as an abstraction of the most important requirements. Scenario view is redundant with other ones, but we use it to identify architectural elements of system and help validating and illustrating the architecture design.

2.2 Quality Attributes of Software Design

2.2.1 Definition of Software Quality

The Institute of Electrical and Electronics Engineers' (IEEE) Standard Glossary of Software Engineering Terminology defines quality as "the degree to which a system, component, or process meets (1) specified requirements, and (2) customer or user needs or expectations." [21] The International Standards Organization (ISO) defines quality as "the totality of features and characteristics of a product or service that bear on its ability to satisfy specified or implied needs." [22]

Below is a list of software quality criteria from "Handbook of Software Quality Assurance" [23]:

- **Correctness**: extent to which a program fulfills its specifications.
- **Efficiency**: use of resources execution and storage.
- **Flexibility**: ease of making changes required by changes in the operating environment.
- **Integrity**: protection of the program from unauthorized access.
- **Interoperability**: effort required to couple the system to another system.

- **Maintainability**: effort required to locate and fix a fault in the program within its operating environment.

- **Portability**: effort required to transfer a program from one environment to another.

- **Reliability**: ability not to fail.

- **Reusability**: ease of re-using software in a different context.

- **Testability**: ease of testing the program to ensure that it is error-free and meets its specification.

- **Usability**: ease of use of the software.

In a real world, it is impossible to get an architecture that meets every criterion. For example, some criteria (e.g. portability) in the list require extra work and codes that would decrease the efficiency. So the architects have to identify the quality attributes of alternative architectures and then make trade-off decision according to the purpose and use of system.

In this project we focus on the criteria of reliability and costs.

2.2.2 Theory of Software Reliability

IEEE defines software reliability as "The probability that software will not cause the failure of a system for a specified time under specified conditions. The probability is a function of the inputs to and use of the system, as well as a function of the existence of faults in the software. The inputs to the system determine whether existing faults, if any, are encountered."[24].So reliability means the probability of the occurrence of observable failure of a system.

A number of techniques are available to calculate the reliability of system, including Reliability Block Diagrams (RBD) [35] and Markov analysis [36]. They use components' reliability, given usage paths information and structural relationships, to predict reliability of the assembly of components.

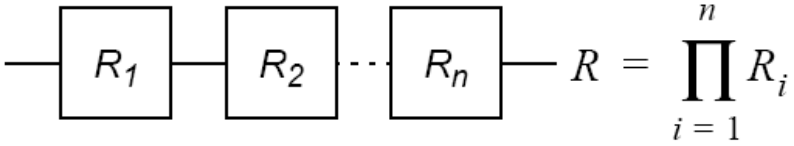
Markov analysis develops the reliability of a component being in a given state. Its advantage is that it describes both the failure of a component and its subsequent repair. It is a powerful reliability tool for analyzing complex, dynamic systems. One known problem in the use of

Markov chains is the rapid growth of the states and complexity [37]. Markov models may contain hundreds of states. The problem can be solved because reliability permits a hierarchical approach.

In my report, we show how reliability block diagrams are used to estimate the reliability of different competing component-based software architectures. First, I begin by presenting a brief background of reliability block diagrams.

A reliability block diagram (RBD) is a graphical depiction of the system's components and connectors, which can be used to determine the overall system reliability given the reliability of its components [35]. An RBD has one or more paths through it, which represent successful system operation. Every path is constructed out of components and connectors. If any path is executed successfully, then the overall system is said to succeed, otherwise if any path fail, then the overall system also fails.

Our component-based system is composed of a number of scenarios, which are chains of components. The overall reliability of such a pipeline can be calculated as the probability of all components executing successfully, or the product of the individual reliabilities.



Even if the reliabilities of all components are known, it is hard to know how a component will affect others, e.g. whether the failure of a component will result in others' failure. We assume that any failure happened on the scenario execution path implies the failure of whole system. One of our assumptions is that the quality of each component in system is independent.

2.3 Quality Exploration for Component-Based Architectures

Our research is in the environment of Robocop Component-Based Architecture (CBA) [19]. The Robocop CBA complies with the common COTS (commercial off-the-shelf) approaches [31] for component-based development. This architecture is developed for middle-ware in consumer devices, with an emphasis on robustness and reliability. The Robocop CBA is similar to Koala [20], but realizes quality attributes more efficiently via modeling techniques. For example, a component designer can supply component reliability and cost data (like data plates on machine) along with the component executable code. A designer composes an application from a number

of components and can predict the overall reliability and total cost of the system, using the set of such available information.

Below we provide some background knowledge about Robocop CBA, related design space exploration method and toolkits. This is the former work done in [1]. For more detail, please refer to [19].

2.3.1 Robocop CBA

The Robocop CBA allows composing software from components. The primary benefit of the component paradigm is that architects can buy the third-party well-defined components from market to build their systems. That will decrease the time and cost of development cycle, and allows a flexible selection of the components to form a composition of them and to get the highest value in the multi-objective quality attribute analysis.

Let us now see what the Robocop component model is to be when adding our reliability and cost models. A Robocop component is a set of possibly related models, as depicted in Fig. 2. Each individual model provides a particular type of information about the component. Models can be represented in readable form (e.g. documentation) or in binary code (exe files). One of the models is the executable model that contains the executable component. Other examples are: reliability model, resource model, cost model, and behavior model.

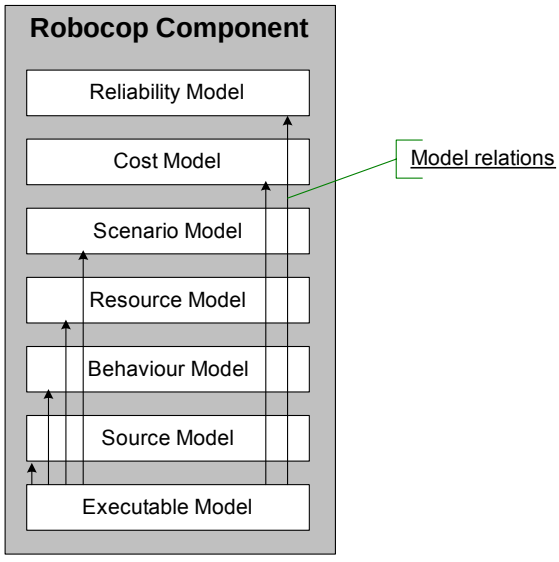
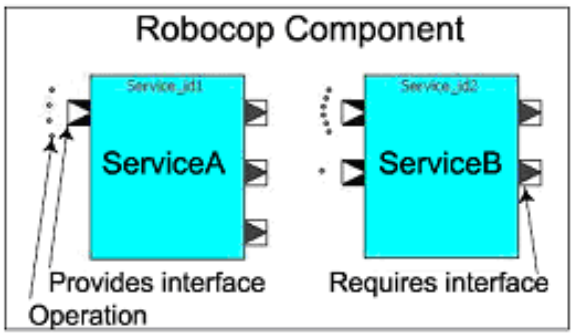


Figure2 Robocop component model from [1]

A component offers functionality through a set of "services" P (see Fig. 3(a)). Services are static entities; they are instantiated at run-time, using a service manager. The resulting entity is called "service instance", which is a Robocop equivalent of an object in OO programming. A Robocop service may define several interfaces (ports). We distinguish a set of "provide" ports PR and a set of "require" ports REQ. The former defines interfaces that are offered by the service, while the latter defines interfaces that the service needs from other services in order to operate properly. An interface is defined as a set of operations.



(a)

m	= *P*, where *m* is an *Executable Model* and *P* is a set of individual *p*'s (services).
p	= *(PR, REQ)*, where *PR* is a set of *pr* (provided ports) and *REQ* is a set of *req* (required ports),
pr	= (*name, interface*),
req	= (*name, interface*),
interface	= *O*, where *O* is a set of *impl_opr* (implemented operations).

(b)

Figure 3 (a) Example of executable component
(b) Specification of a component model. (Source [1])

The binding between service instances in the application is made via a pair of provides-requires interfaces. A service p being part of the above-mentioned executable model is specified in Fig. 3(b).

The Robocop CBA is implementation-independent. The architecture has no limitations on programming languages and platforms. A service can implement any number of threads. Besides this, both synchronous and asynchronous communications are possible.

2.3.2 Design Space Exploration Methodology

During the system design phase, we assume that components are available, and their necessary models (resource, reliability, behavior models) have been specified when they are shipped. The DAX methodology comprises several phases (see Fig.4), which can be iteratively repeated until the best design solution is found. We have developed extensions to a toolkit called Robocop Integration Environment (RTIE) to compose components and simulate the complex processes.

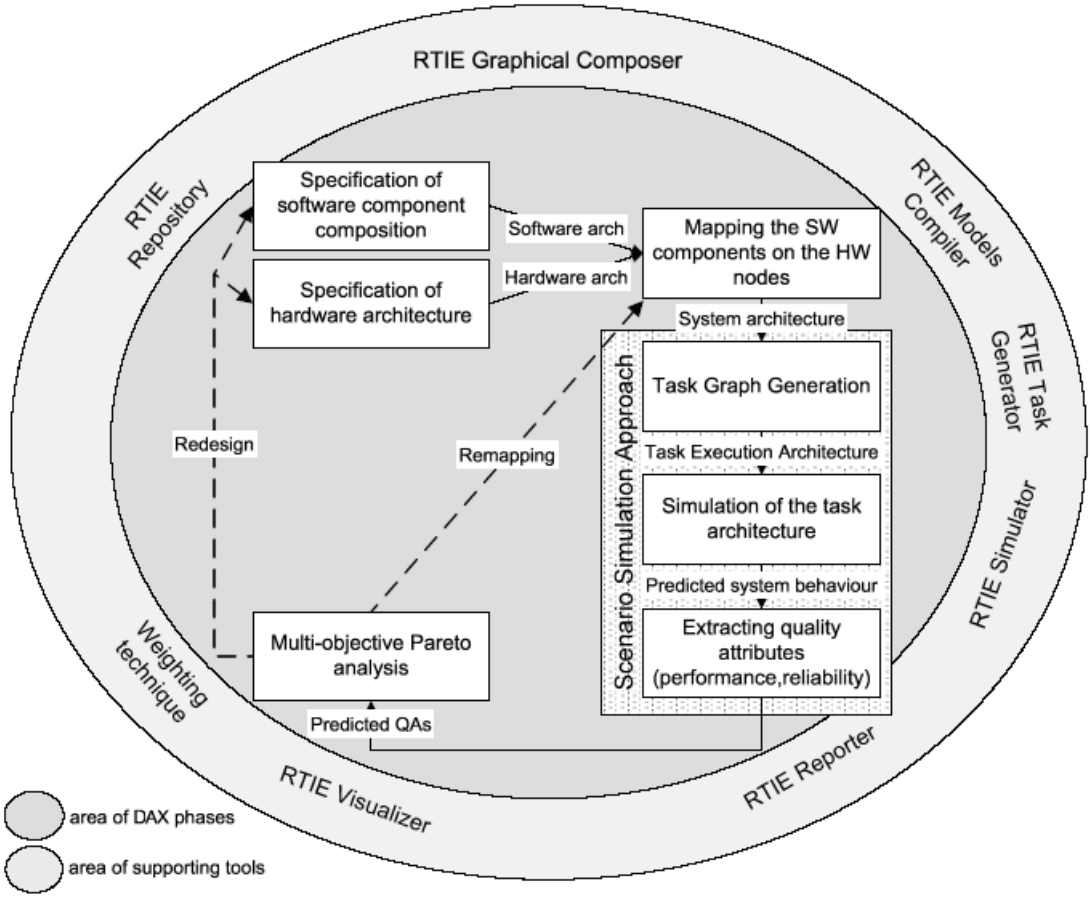


Figure 4 Main Phases and supporting tools of the DAX methodology

After the analysis of system requirements, the designer selects appropriate software components, which are available from the RTIE repository, to compose the software system. We use deployment diagram in UML as hardware specifications.

Except for cost model, we need to know or predict the behavior of system in order to analyze important quality attributes such as reliability. Reliability models of the constituent software and hardware elements are mapped on the predicted dynamic architecture. A reliability model of an

element specifies a probability of failure of that element. A component reliability model describes probability of failure of each operation implemented by the component. Dynamic architecture shows which operations are executed and how many times per defined simulation period each operation was executed. Thereby, aggregating the probabilities of failure for the operations, nodes, memories and physical links, we can calculate system reliability for certain period.

The RTIE Reporter analyses the simulation results and outputs the quality data to the RTIE Visualizer.

All detailed information on the topic is available in [1].

Before the thesis, we already have behavior model and scenario model for task simulation, and resource model for performance prediction. The main contribution of this project is to introduce new quality models (i.e. reliability, cost) and architecture models (e.g. use case, deployment) extending Robocop models and DAX methodology so that the quality analysis ability of RTIE is promoted from scenario-level to architecture-level.

Currently our architecture consists of a set of models, including 4 structural models and 3 quality relevant models. They are jointly compiled by the RTIE Complier. The objective of the composition is to collect task-related data spread over different models and components and then generate the tasks running in the application. For example, the information about the operation call order sequence is spread over component behavior models, whereas the reliabilities of each involved operation and interface are specified in reliability model. More detail about the scenario simulation approach is in [26].

Below we will first introduce a car navigation system as a case study, and then explain the models used in our multi-qualities analysis methodology.

3.1 Car Navigation Case Study

This example is a distributed car radio navigation system selected from the case study for Modular performance Analysis given in [25].

An overview of the system is presented in Figure 5; it is composed of three main clusters of functionality.

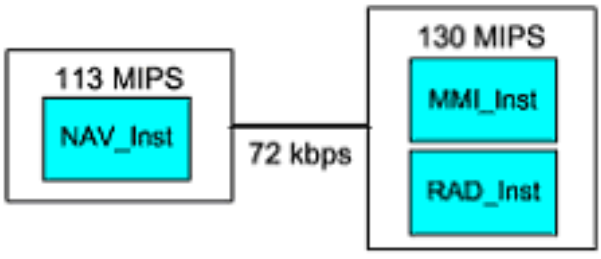


Figure 5 High-level overview of a distributed radio navigation system

– The man-machine interface (***MMI***) which takes care of all interactions with the user, such as handling key inputs and graphical display output.

– The radio functionality (***RAD***), which is responsible for basic tuner and volume control as well as handling of traffic information, services such as RDS-TMC (Traffic Message Channel). TMC is broadcasted along with the audio signal of radio channels.

– The navigation functionality (***NAV***) that is responsible for destination entry, route planning and turn-by-turn route guidance giving the driver both audible and visual advises. The navigation functionality relies on the availability of a map database (typically stored on a CD or DVD) and positioning information (for example GPS).

In our case, the three functionalities are designed as three components separately. They are deployed on two physical nodes, "car" and "service center".

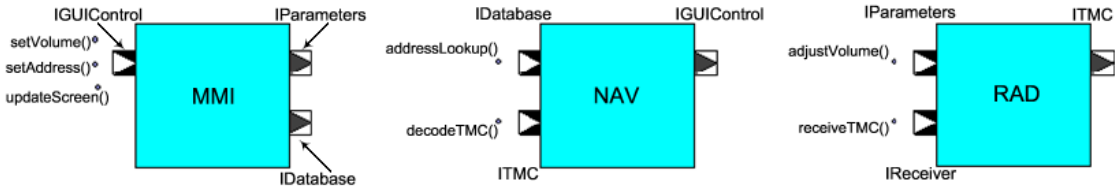


Figure 6 Component used in this example (source [1])

We have two use cases:

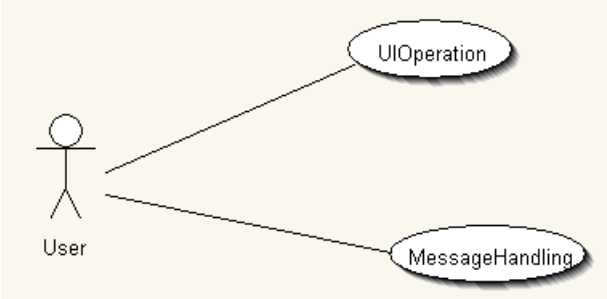
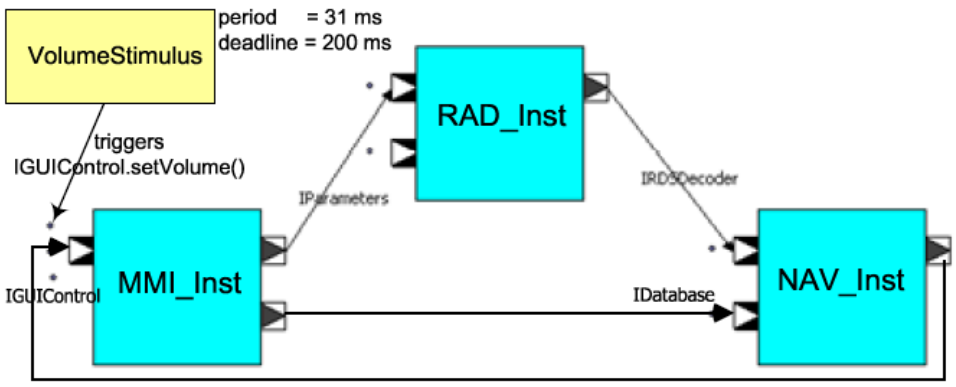


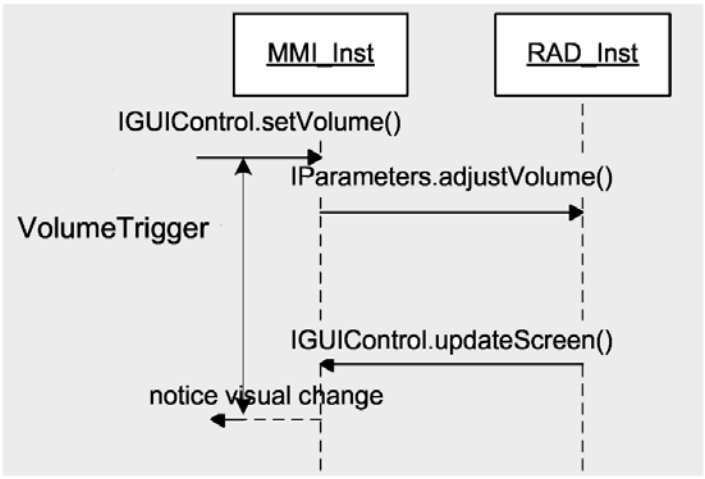
Figure 7 Use Case Diagram

The use case "UIOperation" contains two scenarios:

1. "Change Volume" (see Figure 8) - The user turns the rotary button and expects instantaneous audible feedback from the system. Furthermore, the visual feedback (the volume setting on the screen) should be timely and synchronized with the audible feedback.



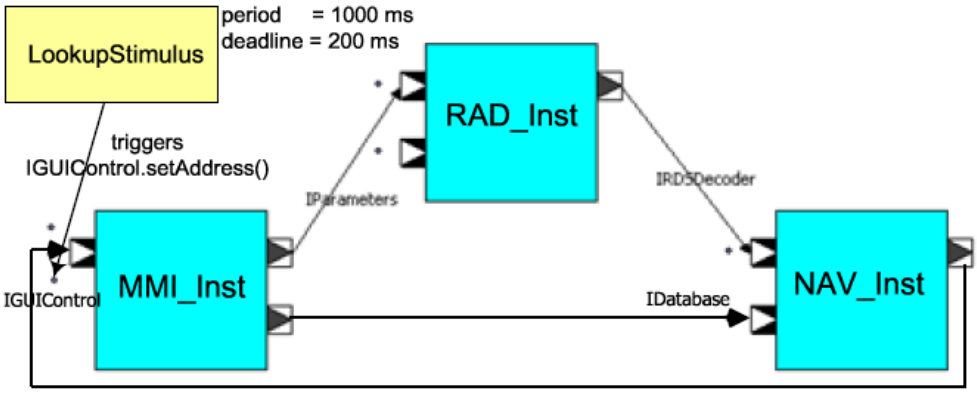
(a)



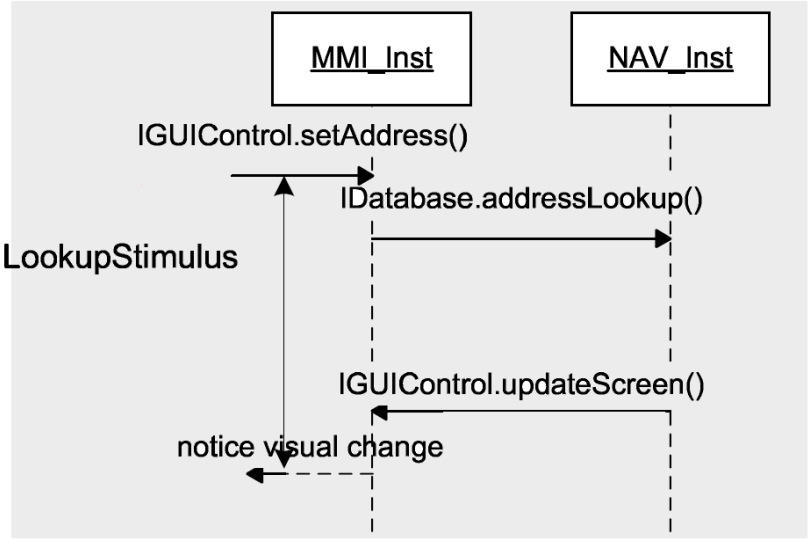
(b)

Figure 8 (a) Graphical Representation of "Change Volume" Scenario
(b) Generated task of the scenario

2. "Address Look-up" - Destination entry is supported by an interactive interface. The display shows the alphabet and the user selects the first letter of a city (or street). By turning a knob the user can move from letter to letter; by pressing it the user will select the currently highlighted letter. The map database is searched for selected letter and then presents the user with a list of fully expanded city (or street) names.



(a)

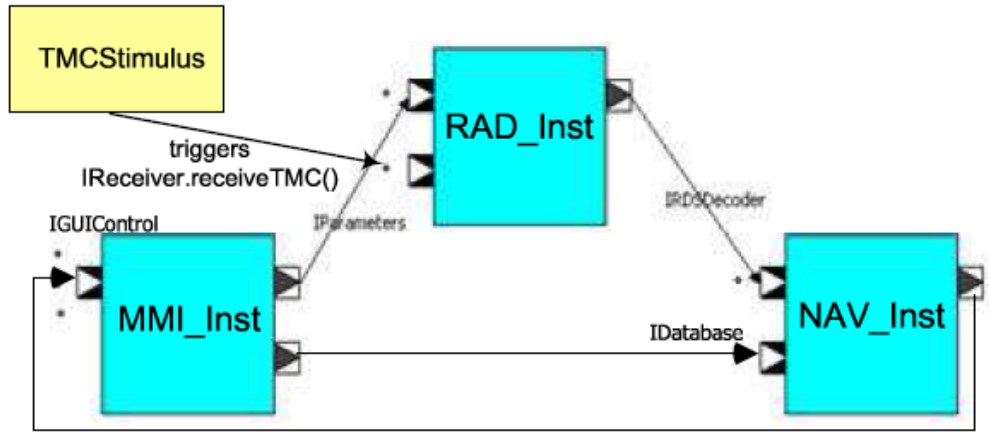


(b)

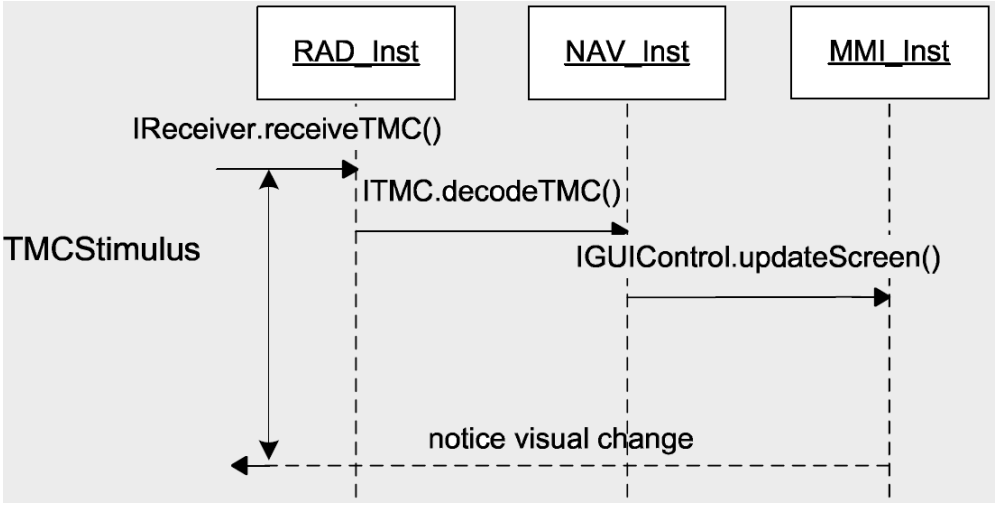
Figure 9 (a) Graphical Representation of "Address Look-up" Scenario
(b) Generated task of the scenario

"Message Handling" use case contains one scenario:

"TMC Message Handling" - Digital traffic information is very important for in-car navigation systems. It enables the features such as automatic re-planning of the planned route if a traffic jam occurs ahead. RDS TMC is a good example of digital traffic information service. TMC messages are broadcasted by radio stations together with stereo audio sound. Traffic information messages are received while the drivers are listening to their favorite radio station. RDS TMC messages are encoded; only traffic relevant messages are transmitted.



(a)



(b)

Figure10

(a) Graphical Representation of "TMC Message Handling" Scenario

(b) Generated task of the scenario

3.2 Structural view models

In Robocop CBA models, we use four models, which describe the static structure and behaviors of system architecture. They are Scenario Model, Behavior Model, Use case Model and Deployment Model.

3.2.1 Scenario Model

For system architecture, the designer defines a set of scenarios and specifies an application scenario model for each of them. In the scenario, the designer specifies task triggers that start a scenario. Before making the architecture of a system, we have a repository of components.

Architects select components from the repository, and connect them by linking each provided interface with a matching required interface. A scenario starts from the trigger that calls the first operation. For a trigger, a designer may define some parameters, for example, the burst rate, period deadline etc. A scenario model defines the binding of interfaces, but the corresponding behavior model determines the operations sequences. Figure 11 is the metamodel of Scenario Model, Figure 9 (a) and Figure 10(a) are two examples of scenario diagrams that are implemented in our tool. In Figure 9 (a), the designer introduces a periodic trigger that calls volume setting operation each 31ms, and specifies the behavior of the scenario.

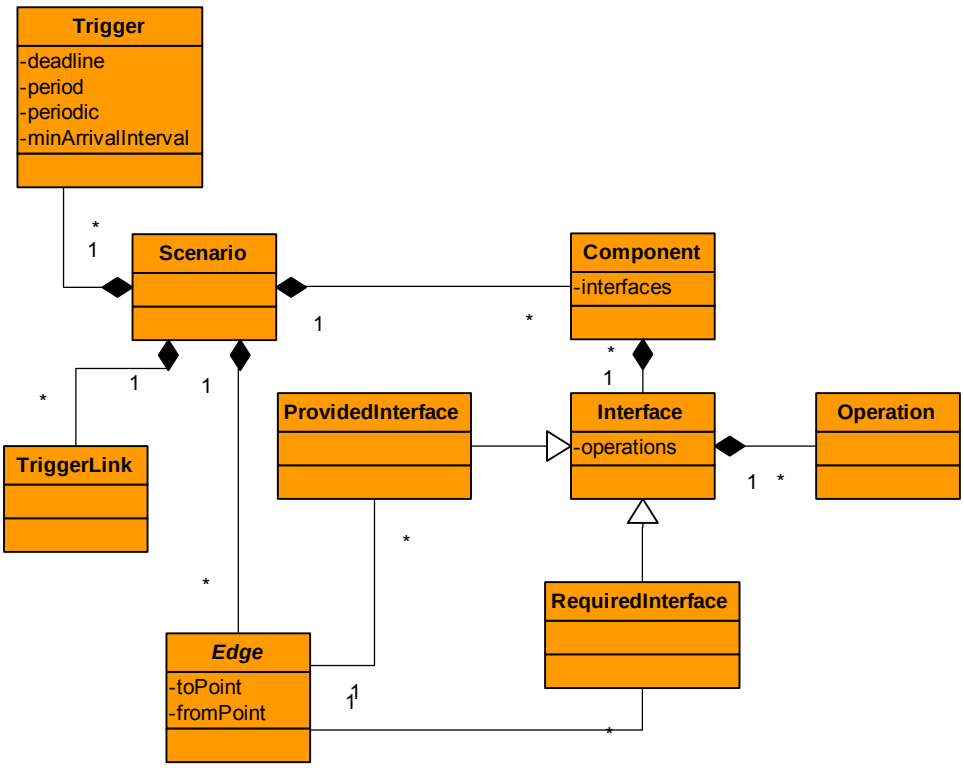
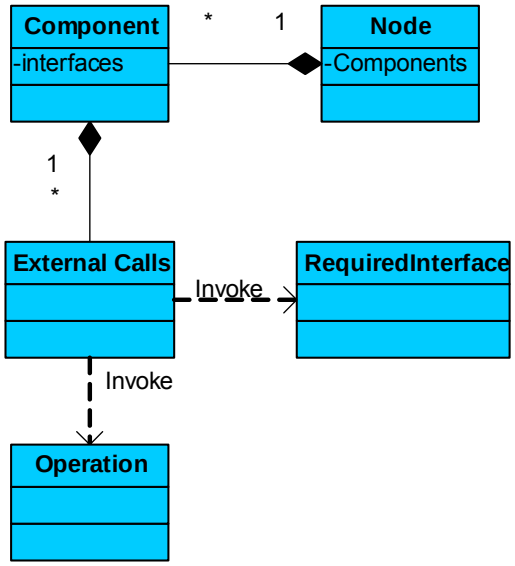


Figure11 Scenario Metamodel

3.2.2 Behavior Model

The behavior model relevant here specifies for each operation of a component a sequence of external calls to other interfaces' operations. The external call according to [1] is a usual method invocation made inside the implemented operation. Behavior models only describe the links between operations, while scenario models represent components and connections between their interfaces.

Figure 12 (a) is the metamodel of behavior model. An example of a behavior model in Altova XMLSpy 2005(evaluation version) [28] is shown in Fig 12 (b). It describes operations and their external calls present in scenario "Message handling" (see Figure 10).



(a)

	operationName	fromInterface	fromServiceName	behaviour
1	receiveHandleTMC	IRDSReceiver	Radio	behaviour calledOperation name: decodeTMC fromInterface: IRDSDecoder fromService: {00000000-0000-0000-0000-000000000003} synchronousCall: true numberOfIterations value: 1
2	decodeTMC	IRDSDecoder	Navigation	behaviour calledOperation name: updateScreen fromInterface: IOutput fromService: {00000000-0000-0000-0000-000000000002} synchronousCall: true numberOfIterations value: 1
3	updateScreen	IOutput	UserPanel	

(b)

Figure12 (a) Behavior Metamodel (b) Example of Behavior model

3.2.3 Use case Model

The use case model here is derived from the use case view of "4+1" models, which has already been explained in chapter 2. It is the abstraction of most important functional requirements in system architecture. The difference between the use case models in UML and our model is that scenarios are contained in our use case. Each use case in our model is composed of several scenarios. The occurrence probability of each scenario in one use case is specified. We also have the arrival rate of each use case. The data is used to predict how many errors may happen in a certain amount of time. The concrete calculation algorithm is discussed in chapter 4. Figure 13 is the metamodel of Use case model. Figure 7 is a Use case example.

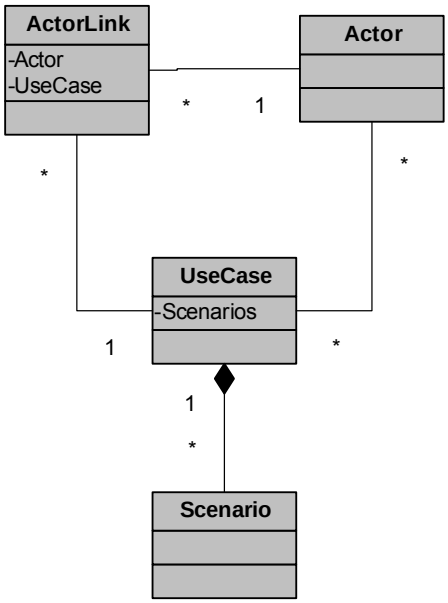


Figure 13 Use case Metamodel

3.2.4 Deployment Model

The deployment model represents the mapping from software components to physical components. Physical nodes are connected by physical links (e.g. cable, telecommunication). Both of them have reliability attributes. Figure 14 represents the metamodel of Deployment model. Figure 5 is a deployment example.

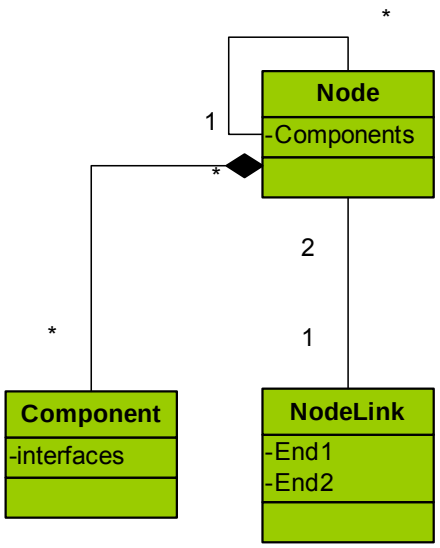


Figure14 Deployment Metamodel

3.3 Quality models

Currently we have three quality metamodels which are respectively related to software system i.e. reliability, performance and cost. We use the quality information of software components and hardware components to predict the quality of the overall system.

3.3.1 Reliability Model

We assume that the reliability of the constituent software and hardware elements is given.

From the software point of view, we need to know the reliability of each operation. These data should be provided by the component provider, like characteristics of a machine. Figure 15(a) represents the reliabilities of the operations called in scenario "Message handling" (see Figure 10).

We also need to consider the reliability of the devices and physical links between them (e.g. cable). The arrival rate of a use case and occurrence probabilities of its scenarios are needed to predict how many errors will happen in a certain time unit. Figure 15(b) contains these kinds of reliability information of the car navigation case study (see Section 2.4).

Figure 15(c) is the metamodel of Reliability model. Chapter 4 tells more details about reliability algorithm.

	operationName	fromInterface	fromServiceName	operationReliability
1	receiveHandleTMC	IRDSReceiver	Radio	operationReliability: value 0.96
2	decodeTMC	IRDSDecoder	Navigation	operationReliability: value 0.94
3	updateScreen	IOutput	UserPanel	operationReliability: value 0.92

(a)

Node (2)

	nodeName	nodeID	nodeReliability
1	Car		0.99
2	Nav		0.99

NodeLink

nodelinkID	
nodelinkReliability	0.99

nodeEnd (2)

	nodeendName
1	Car
2	Nav

Usecase (2)

	ucName	arrivalRate	Task
1	UIOperation	5	Task (2)
2	MessageHandling	10	Task (1)

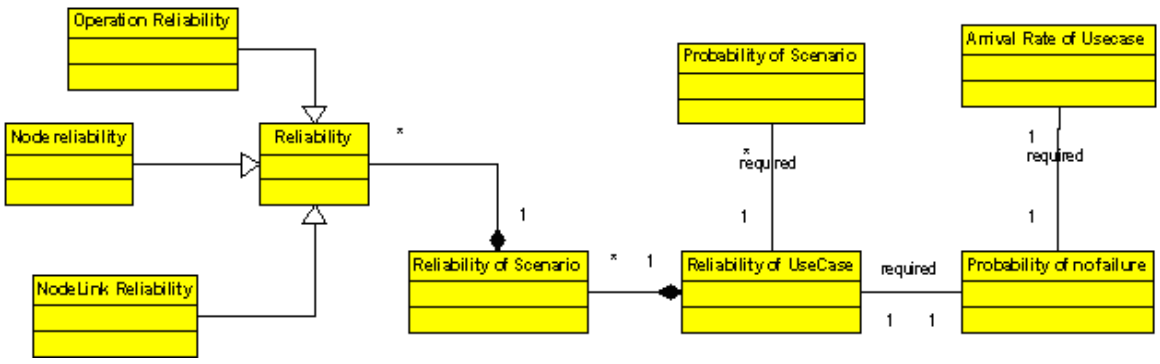
Task (2)

	TriggerName	scenarioProbability
1	VolumeTrigger	0.8
2	Lookup_Trigger	0.2

Task (1)

	TriggerName	scenarioProbability
1	MessageHandling	1.0

(b)



(C)

Figure15 (a) operation reliabilities

(b) other relevant data in reliability model (c) Reliability Metamodel

3.3.2 Resource Model

We assume that each component has already its own resource model. The resource model according to [1] "contains processing, bandwidth and memory requirements of each operation implemented by the component". Figure 16 is an example model, which contains resource information of operations involved in scenario "Message handling" (see Figure 10).

	operationName	fromInterface	cpuUsage	memoryUsage	busUsage
1	receiveHandleTMC	IRDSReceiver	cpuUsage averageClaim 38	memoryUsage averageClaim 20 averageRelease 40	busUsage averageClaim 2000 averageTime 5
2	decodeTMC	IRDSDecoder	cpuUsage averageClaim 192	memoryUsage averageClaim 20 averageRelease 40	busUsage averageClaim 2000 averageTime 5
3	updateScreen	IOutput	cpuUsage averageClaim 19	memoryUsage averageClaim 20 averageRelease 40	busUsage averageClaim 2000 averageTime 5

Figure 16 Example of Resource model

3.3.3 Cost Model

We assume that prices of the software and hardware constituents of the system are known. The cost of the overall system is calculated by adding all the constituents' prices together. Such an example of Cost model is represented in Figure 17 (a). The metamodel of cost model is shown in Figure 17 (b).

Component (3)

	Name	compID	cost
1	Radio		300
2	Navigation		5000
3	UserPanel		200

Node (2)

	Name	nodeID	cost
1	Car		20000
2	Nav		100000

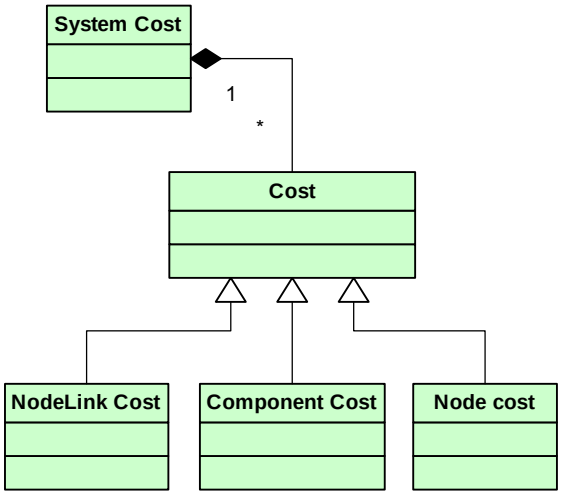
NodeLink

nodelin...	
cost	10000

nodeEnd (2)

	nodeendName
1	Car
2	Nav

(a)



(b)

Figure17

(a) Example of Cost model (b) Cost Metamodel

4. Analysis Algorithms

4.1 Reliability Calculation

In this paper, we introduce a methodology to explore and analyze the reliability of system architectures. It is a path-based early reliability assessment [27], but adapted to the Robocop component-based system environment. We use data from the behavior, deployment and reliability models in order to predict the overall reliability. But our reliability assessment approach is quite different with the one in paper [27]. That paper counts the reliability of software components into the reliability calculation of overall system, which we don't count. But it skips the reliability of the physical nodes and the operations of components. In other words, paper [27] assumes that each executing of the components has the same reliability; it also assumes that each physical node is absolutely reliable (100% reliability).

One of the most important assumptions in our project is that we already have the reliability information of each software and hardware element of the system. Using this data, we are able to predict the fail rate of the architecture at the early phase of lifecycle. Reliability of the components and devices may be obtained from prior testing, and reused in our analysis. We consider all possible program execution paths with frequencies, and their computed reliabilities as a basis for reliability calculation. Paths are simulated and derived from the behavior models and the scenario models. Probabilistic techniques for reliability are used at the design-level, considering software components, connectors and physical nodes. As a result, architects can select elements to compose a system that fits reliability requirement(s).

4.1.1 Use of Sequence Diagrams

In our tool, sequence diagrams are generated by simulation of components behavior and scenario models. They describe the interaction between components in order to achieve a given scenario. A scenario is a set of ordered operations. In Figure 18, a sequence diagram is shown:

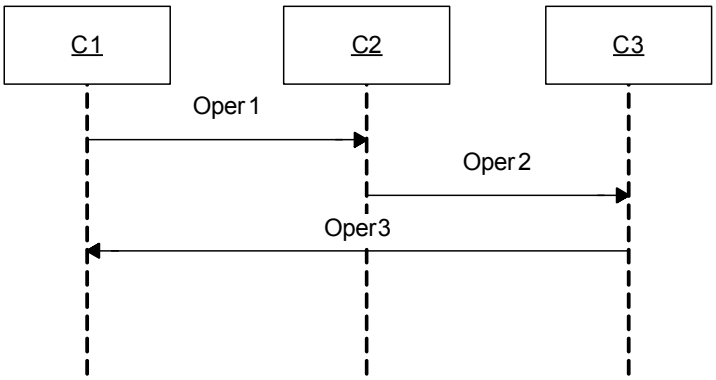


Figure18: Annotated Sequence Diagram

The scenario is composed of a sequence of operations. The overall reliability of such a sequence can be calculated as the probability of all operations executing successfully, or the product of the individual reliabilities. Similar calculation algorithm refers to [35].

The reliability of this scenario j is given by:

$$\varphi_j = \prod_{i=1}^{N_j} r_i^{om_{ij}} \qquad (1)$$

Where om_{ij} is the frequency of operation *i* in scenario *j*, r_i is the success probability of operation *i*, N_j means the number of operations in scenario *j*.

4.1.2 Use of Deployment Diagrams

A deployment diagram shows the software components distribution over physical devices. Nodes represent devices where the components are running on (e.g. server, PCs, etc.) and links represents hardware connectors (e.g., cable, wireless, etc). Software components are placed into the respective nodes. Figure19 shows an example of a deployment diagram including components in Figure18.

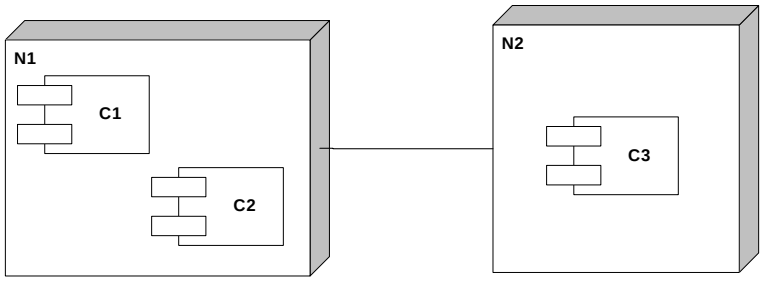


Figure 19: Annotated Sequence Diagram

Each node and node link has its reliability information. The failure of a node or link will result in the failures of related operations. We assign the link reliability to each interaction of a sequence diagram. If two interacting components are located in the same node, the reliability of their interaction is 1.0. If there is no link between the two components, their connection reliability is 0.0.

The node reliability also needs to be considered when we calculate the reliability of its loaded operations.

We have a new equation (2) which allows reliability of nodes and connections to be used in the reliability predication for scenario j. Every path of an operation executing is constructed of the called operation, the node where the operation is deployed and the node connector which is on the operation executing path. All these elements are essential to the success of the operation. If all elements are executed successfully, then the operation is said to succeed, otherwise if any element fails, then the operation executing also fails. Therefore, the probability of an operation executing successfully is the product of the reliability of this operation, reliability of its deployed node and reliability of its path (if the caller operation is not on the same node). Mappings from software components to physical nodes are specified in our deployment models.

$$\varphi_j = \prod_{i=1}^{N_j} (r_i \times rn_i)^{om_{ij}} \bullet \prod_{(m,n)} \psi_l^{|Interact(m,n,j)|} \qquad (2)$$

Where om_{ij} is operation i's frequency in scenario j, r_i is the reliability of operation i, rn_i is the reliability of the node where the operation i is running, N_j means number of operations in scenario j, ψ_l is the reliability of connector l, $|Interact(m,n,j)|$ means the number of physical interactions that components m and n exchange in the scenario j [27]

Deployment diagram is a mapping of software components to nodes. If we have several mapping alternatives, our methodology will provide reliability analysis results of each alternative.

4.1.3 Use of the Use Case Model

A Use Case diagram describes scenarios in a system and how the external user (i.e. actor) interacts with the system. As we described in Chapter 3, each use case contains a set of tasks that are called scenarios.

In the simple Use case example below (figure 20), an actor operates on two use cases. The annotations introduced are explained below: λ_1 and λ_2 represent the occurrence times in a time unit (e.g. one hour) that the use case uc1 or uc2 are executed respectively; sP11 and sP12 represent the occurrence probabilities of scenario s11 or s12 respectively when use case uc1 is requested, sP11 plus sP12 equals 1.0; sP21 and sP22 have similar meaning with respect to use case uc2.

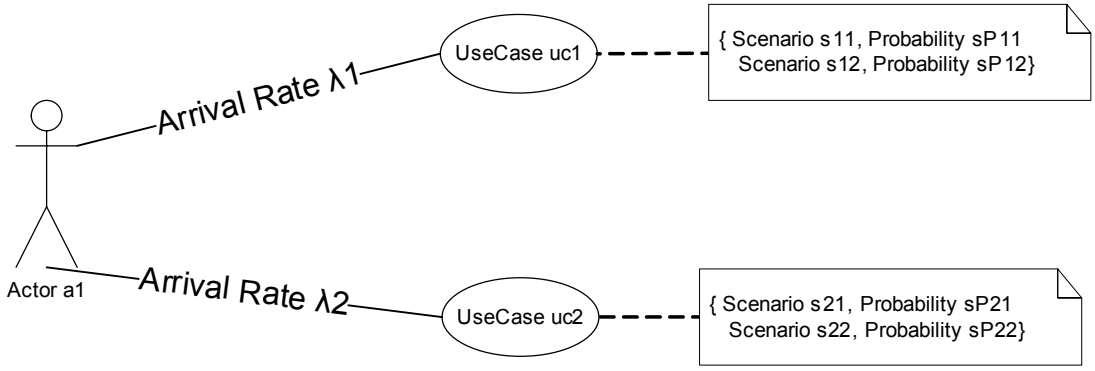


Figure20: Annotated Use Case Diagram

We know that each use case is made of a set of scenarios, and each scenario is specified in a non-uniform probability distribution to the same use case. Therefore, the reliability of executing the use case u is given by:

$$\theta_u = \sum_{j=1}^{K_u} P_j \varphi_j \qquad (3)$$

Where P_j is the probability of scenario *j* executing, φ_j is the reliability of scenario j, K_u is the number of scenarios in use case *u*.

We get the estimate of the probability γ of no failure for a use case *u* in the period of *[0...T]* from the following equation (4):

$$\gamma_{uT} = \theta_u^{\lambda_u T} = (\sum_{j=1}^{K_u} P_j \varphi_j)^{\lambda_u T} \quad (4)$$

Where θ_u is the reliability of use case *u*. Thus if the reliability of a use case is 99.99%, it is estimated to have an arrival rate of 10 executions per hour, then its reliability over a 24 hour period is calculated to be approximately 97.6%.

4.1.4 Whole system formula

In order to get the reliability prediction formula of the whole system, we combine the above equations together and form one equation (5) as follows:

$$\gamma_{uT} = \left\{ \sum_{j=1}^{K_u} P_j \left[\prod_{i=1}^{N_j} (r_i \times rn_i)^{om_{ij}} \bullet \prod_{(m,n)} \psi_l^{|Interact(m,n,j)|} \right] \right\}^{\lambda_u T} \quad (5)$$

A system may contain more than one use cases. These use cases execute simultaneously. In a period of time, only if any use case is executed successfully every time, the overall system is said to succeed during the certain time slot. Otherwise if any use case fails, then the overall system also fails. The failure times of overall system are the accumulation of the individual use case failure times. Thereby the reliability of overall system over a period of time T is the product of the reliabilities of individual use cases during the time slot.

We can derive the reliability of the system over a period of time T using the following equation:

$$R_T = \prod_{u=1}^{M} \gamma_{uT} \quad (6)$$

Where γ_{uT} is the reliability of use case u over the period of time T, M means the number of use cases in the system.

Equation (5) and (6) represent the mathematical formulation of our reliability model, and will be applied in following two case studies.

4.2 Cost Calculation

We assume that the price of each component and physical device (i.e. node and connector) has been specified. In that case, we just need to sum their prices and then get overall cost of the system. The method can be used to predict the cost of each architecture alternative and make a compromise between it and other quality attributes (e.g. reliability).

4.3 Performance Simulation

We assume that designers know execution time of each operation. Following the DAX methodology, we simulated the execution of the defined scenarios on system architecture. [1] Before simulation the tool performs preprocessing of the computation and communication time data. For each of the processing node, the execution times of all operations to be executed on the node are calculated from the component resource and node performance models (execution time = CPU claim value * processor speed) [1]. The communication time of the operation calls made through processors' boundaries is calculated by dividing the bus claim value of an operation on bus bandwidth value, defined in a performance model of the bus. The scenario simulation results in predicted time consumption of each scenario. We can check if the architecture alternatives satisfy given real-time requirements and which of them has better performance.

4.4 Analysis of the Car Navigation Case

We demonstrate the approach through the case study given in Chapter 3.

Two use cases are shown in Figure7. For each use case a set of scenarios are designed. For example, the use case "UIOperation" has scenario "Change Volume" and scenario "Address Look-up". Each scenario is a task that is composed of a set of ordered operations. The scenario task "Address Look-up" is shown in Figure 9 (b): it is a sequence of three operations, which belong to two different interfaces (ports) and two different components.

The values given to the task "Address Look-up" are the following:

Operation reliabilities:

Operation	fromInterface	fromComponent	Reliability	annotation
setAddress	IGUIControl	MMI_Inst	0.99999995	r_1
updateScreen	IGUIControl	MMI_Inst	0.9999990	r_2
addressLookup	IDatabase	NAV_Inst	0.99999995	r_3

We also need to consider the reliability of physical nodes and connectors, which will affect the running operations. In a deployment diagram (see Figure 5), we know that the component MMI_Inst is deployed in node Car, while NAV_Inst is deployed in Service Center. They are connected through a link. The values set in related reliability model are the following:

Node reliabilities:

Node	Reliability	annotation
Service Center	0.9999999	rn_1
Car	0.999999	rn_2

The reliability of link *1* that connects the two above nodes is 0.9999999, its annotation is ψ_1.

According to equation (3) we can now formulate the reliability for the scenario 1 (i.e. "Address Look-up") as follows:

$$\varphi_1 = \prod_{i=1}^{N_1} (r_i \times rn_i)^{om_{ij}} \bullet \prod_{(m,n)} \psi_l^{|Interact(m,n,1)|}$$

$$= r_1^{rn_2} \bullet r_2^{rn_2} \bullet r_3^{rn_1} \bullet \psi_1^{|2|}$$

$$= 0.9999966$$

In a similar way, we get the reliability of another scenario in the use case "UIOperation", i.e. for "Change Volume" it is equal to 0.9999996.

We assume that the probability distribution of scenario "Address Look-up" of the use case is 3%, while scenario "Change Volume" has 97% probability distribution. According to equation (2), the reliability of use case "UIOperation" is equal to (0.03 × 0.9999966) + (0.97 × 0.9999996). The result is 0.99999951.

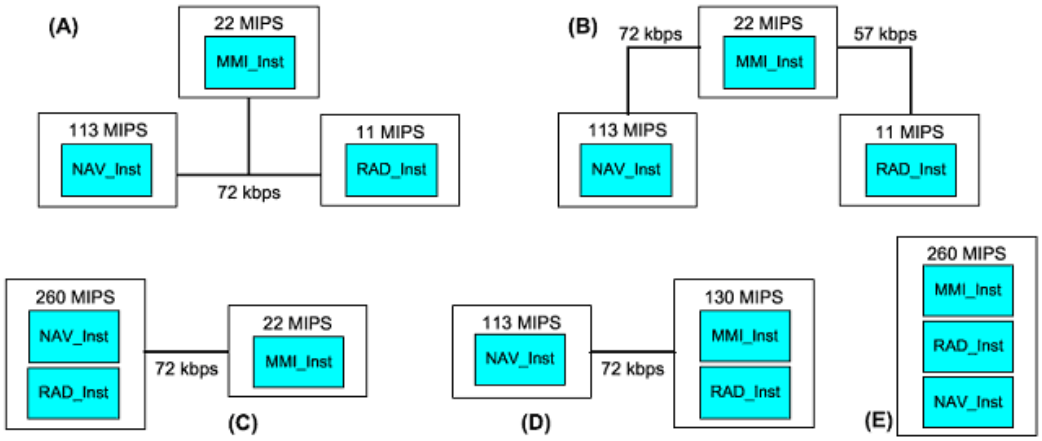
In a similar way, we get the reliability of the use case "Message Handling". The result of formulation is 0.9999973.

We assume that use case "UIOperation" is requested 118800 times per hour, while use case "Message Handling" is requested 1200 times per hour. According to equation (1), the probability of non-failure for the whole system in one hour is equal to $(0.99999951)^{118800_{ij}} \bullet (0.9999973)^{1200_{ij}}$. The result approximately reaches about 94%, which means that the system has 94% probability to be error free during one hour running. The result also indicates that one error happens averagely every 16.67 hours, or after accomplished 2 million tasks (i.e. scenarios).

The system cost attribute was calculated as a cumulative cost of the system hardware and software blocks.

We admit that the reliability and price data for the component and hardware models has been set hypothetically. We did not carry out tests on the components or hardware parts to obtain real data. We correlated the reliability of an operation with its source code complexity.

In paper [25], five alternative architectures are designed with different mapping schemas. The multi-objective design space analysis was carried out against these five alternative solutions with our RTIE tool. Below are the comparisons between alternatives from [1].

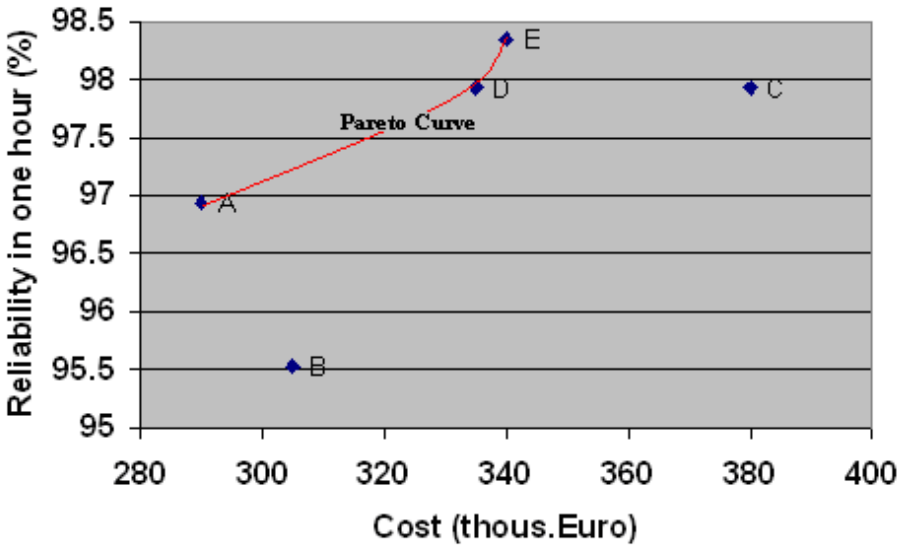


Attribute	Arch. A	Arch. B	Arch. C	Arch. D	Arch. E
Reliability	96.94%	95.53%	97.93%	97.93%	98.35%
Probability of failure	3.06%	4.47%	2.07%	2.07%	1.657%

Calculated reliability values for each architecture.

Attribute	Arch. A	Arch. B	Arch. C	Arch. D	Arch. E
Cost, euro	290	305	380	335	340

Calculated cost values for each architecture.



5. Case Study

This paper presents an iTV system as another case study. It is extended from the first successful field experiments of interactive TV in the Netherlands in December 2001, as stated in [29]. It is proved to be very robust, and recommended for future iTV. Its main functionality (i.e. a nation science quiz) is one of the scenarios in our case. Our case study makes use of its architecture. We should admit that our case is not a real system, so the data used in the case is set hypothetically. Our data is obtained according to the experience of iTV experts from LogicaCMG. This data may not be very accurate but reasonable enough as a case study to show our quality exploration methodology.

5.1 Interactive TV

The experimental iTV architecture described in [29] is designed for supporting a play-along format TV program where the viewers answer the questions via cable (modem) return channel. After reception of the answers and statistics computations on a server, the actual answer statistics of the joint set of participants are transmitted at a specific time setting. We use the hardware in the original system and add some software packages (components) to support new features and services. The architecture diagram of our system is shown in Figure 21.

Set-top box is the most common type of digital TV receiver [30]. It receives digital TV broadcasts from a cable, satellite or terrestrial network, decodes them and then output them to a television. Sometimes it also executes applications included within the broadcast. It contains some software packages including application programs and underlying software components (e.g. drivers, protocols, etc.). We see them as one big single software component in our case. However, Conditional access (CA) module is considered as a separate component, for exploring interactions between components, which are on the same nodes. It is a security software package for pay-TV or pay-Service.

A head end receives the analog signal and converts it into digital stream [30]. It first receives a signal, demodulates the signal, performs first level of error correction, and then turns it to digital stream (MPEG-2). The multiplexer takes one or more MPEG streams, which the STB is interested in, converts them into a single transport stream, and then passes those to other parts of our system. Two examples for that: transmitting data streams to the database or sending audio and video to television.

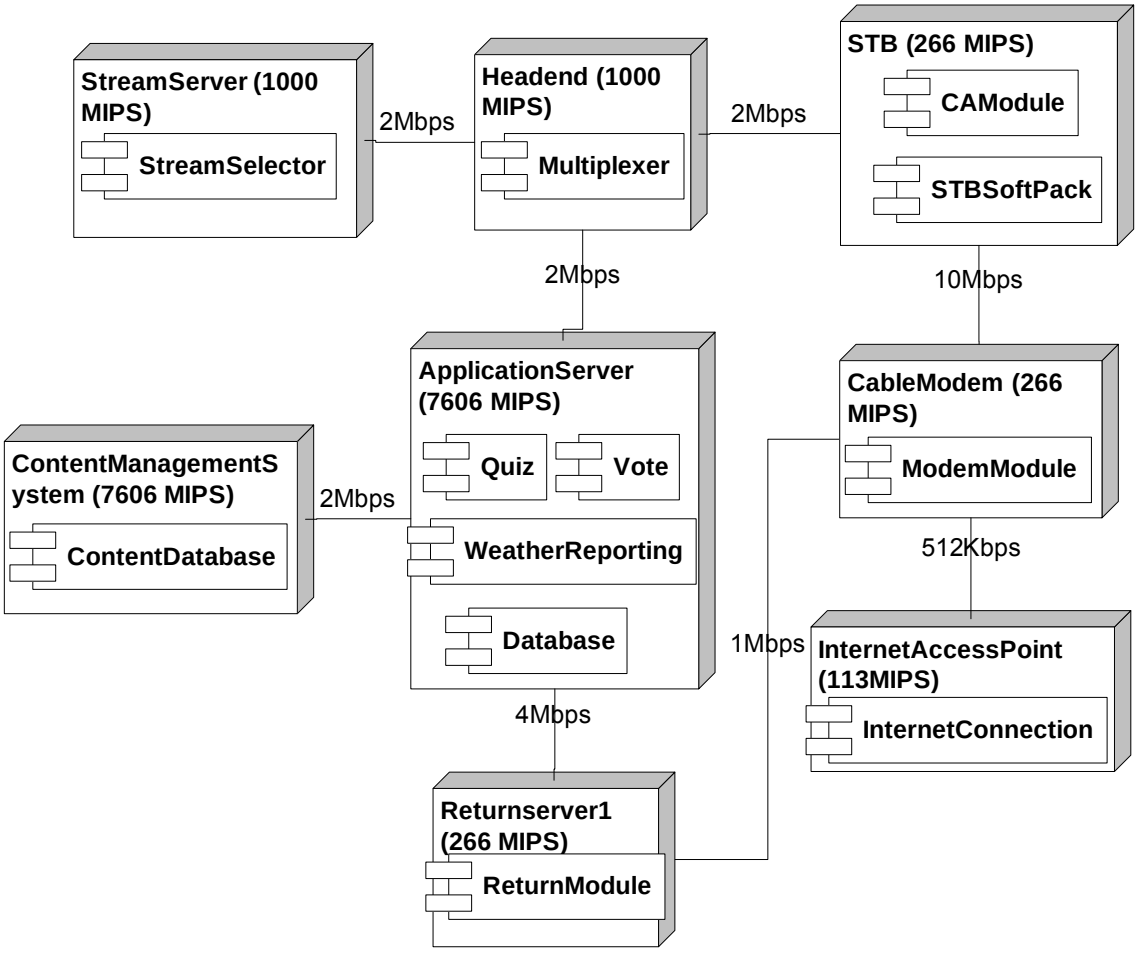


Figure 21: iTV Architecture

A cable Modem is connected to an STB and enables users to interact with TV program or access Internet. According to [29], the stream server delivers the TV program as an MPEG-2 audio/video stream. This stream is multiplexed in the head end with the embedded iTV application. The STB executes the application and initiates a connection with a return server. The user's request is relayed to the content management system through an application server. The content management system generates statistics and then transfers them back to STBs through application server and multiplexer. Internet services are accessed by Cable Modem and Internet Access Point.

5.2 Scenarios and Simulation

In our case, interactive television offers:

- Internet services
- Information services
- Interactive services
- TV Show Services.

The four kinds of service are represented in our use case diagram (figure 22) respectively:

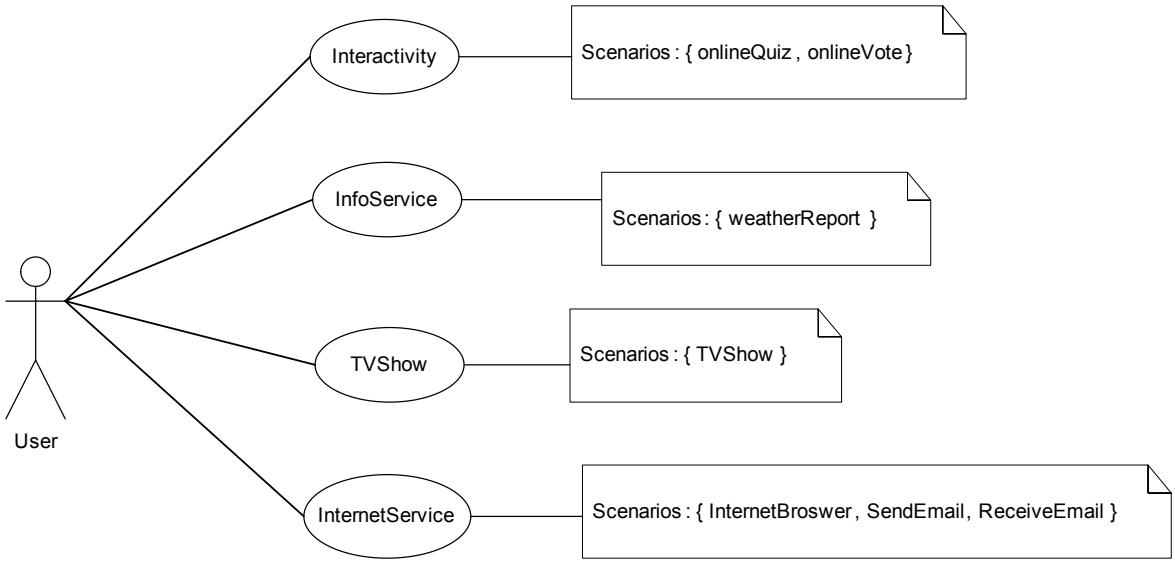


Figure 22: Use Case Diagram of iTV

Each use case contains some scenarios which are described in the linked notations in the use case diagram: the use case "Interactivity" contains for example two scenarios: online Quiz and online Vote.

5.2.1 Use case of Interactivity

The use case "Interactivity" contains two scenarios: online Quiz and online Vote.

Scenario of "online Quiz":

The iTV users participate in the quiz program by providing their answers to the questions via STB and a cable modem. Set-top box deliver the answer to Quiz application component through return server. Quiz application writes the data received in a shared database management system. It thereby accesses the database, wraps the answers in an XML format and forwards them to the content management system. The content management system calculates statistics and sends them back to Quiz application component. Then Quiz component transfers them to a multiplexer. Statistical figures are fed back to the broadcast stream. This enables all participants to see their ranks against other players. The detailed task implementation is shown in Figure 23:

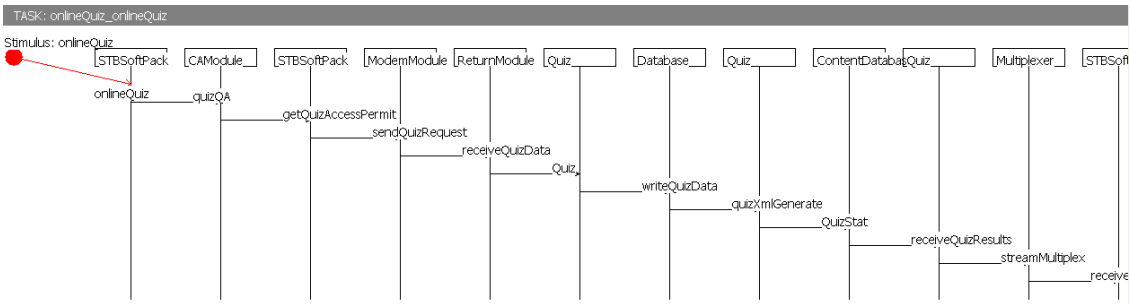


Figure 23: Generated task of Scenario online quiz

Scenario of "online Vote":

The running process of this scenario is quite alike online quiz. They are both interactive services but they use different application component. We replace the Quiz component with vote software package to enable task online vote. The scenario diagram is shown in Figure 24:

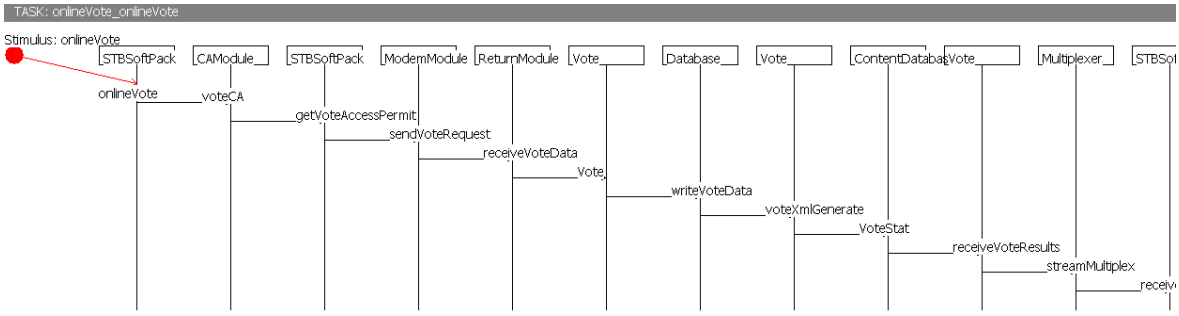


Figure 24: Generated task of Scenario online vote

5.2.2 Use case of Information Service

This use case contains one scenario: weather report.

Users get real-time weather information by sending request to a content database via STB and cable modem. Set-top box delivers the request to weather reporting application component through return server. The application then retrieves weather information from Content Database. Upon reception of the data, it is transferred to a multiplexer. Weather figures are fed back into the broadcast stream and finally shown on television. The scenario diagram of weather report is shown in Figure 25:

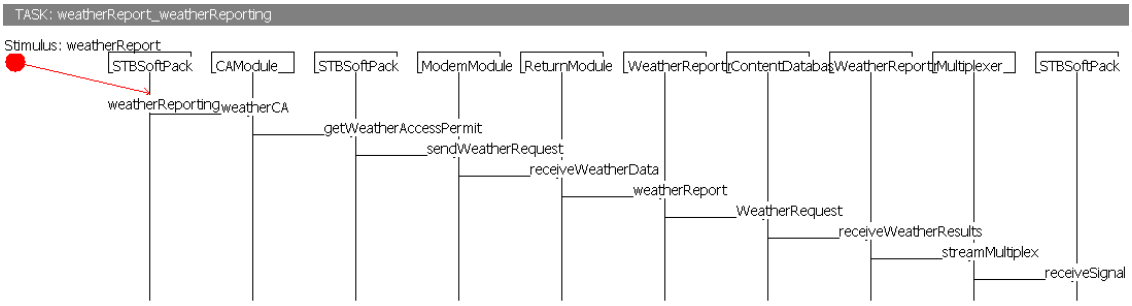


Figure 25: Generated task of Scenario weather report

5.2.3 Use case of Internet Service

Use case "Internet service" contains three scenarios: Internet Browser, Send Email and Receive Email.

Scenario of "Internet Browser":

Users access and browse websites via STB, cable modem and Internet Connection. Set-top box deliver the request to Internet through Internet access point. Upon reception of the data, the data is transferred directly to cable modem and shown on television. The scenario diagram of Internet Browser is shown in Figure 26:

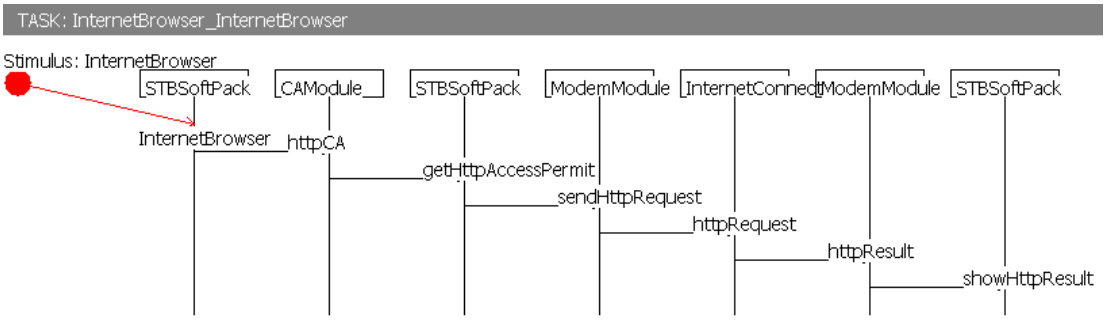


Figure 26: Generated task of Scenario Internet browser

Scenario of "Send Email":

Users send email via STB, cable modem and Internet Connection. Set-top box deliver the email data to Internet through Internet access point. The scenario diagram of Send Email is shown in Figure 27a and 27b:

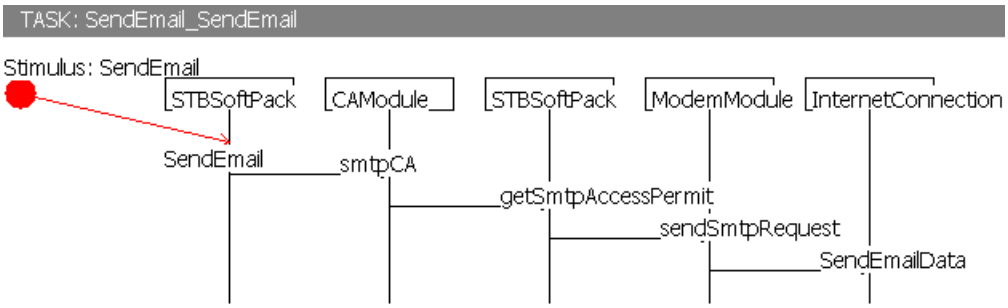


Figure 27: Generated task of Scenario Send Email

Scenario of "Receive Email":

Users send request of checking mailbox and then retrieve mails via STB, cable modem and Internet Connection. Set-top box deliver the request to Internet through Internet access point, and then receive new mails from mailbox. The scenario diagram of Receive Email is shown in Figure 28:

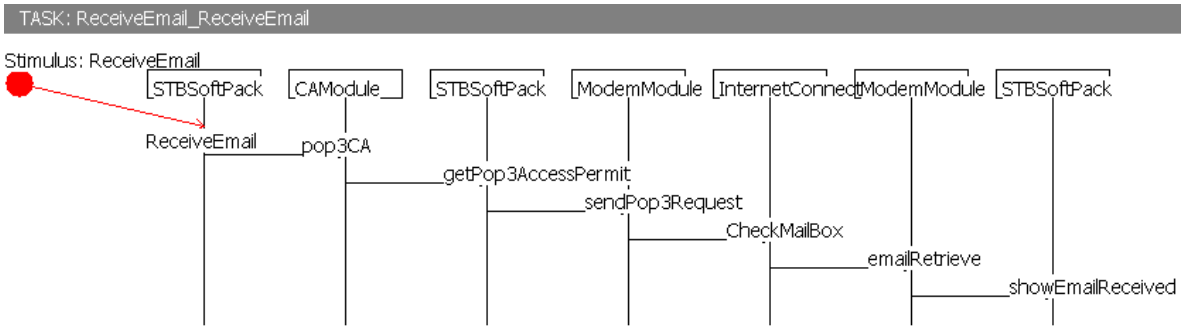


Figure 28: Generated task of Scenario Receive Email

5.2.4 Use case of TV Show

This use case contains one scenario: TV Show.

Scenario of "TV Show":

"iTV system" receives digital TV broadcasts from a cable, satellite or terrestrial network, decodes them and then output them to a television. The scenario diagram of TV Show is shown in Figure 29:

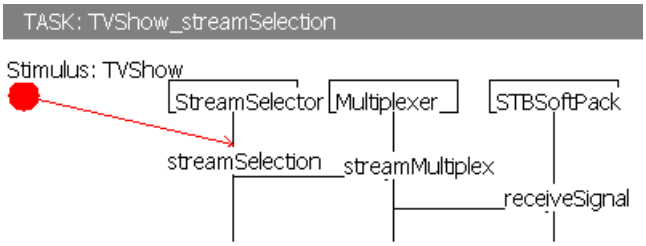


Figure 29: Generated task of Scenario TV Show

5.3 Alternative Architecture

We study another alternative architecture with different mapping schemas (see Figure. 30). It was found that the convergence of all cable modems into a single return server provided a bottleneck for large-scale user access in a short interval. [29] Figure 30 shows set-boxes connecting to one of the available return servers. Each STB randomly selects an available server, thereby delivering a load-balancing solution.

Another change to the original architecture is that we separate database from the application server and deploy it on an individual database server.

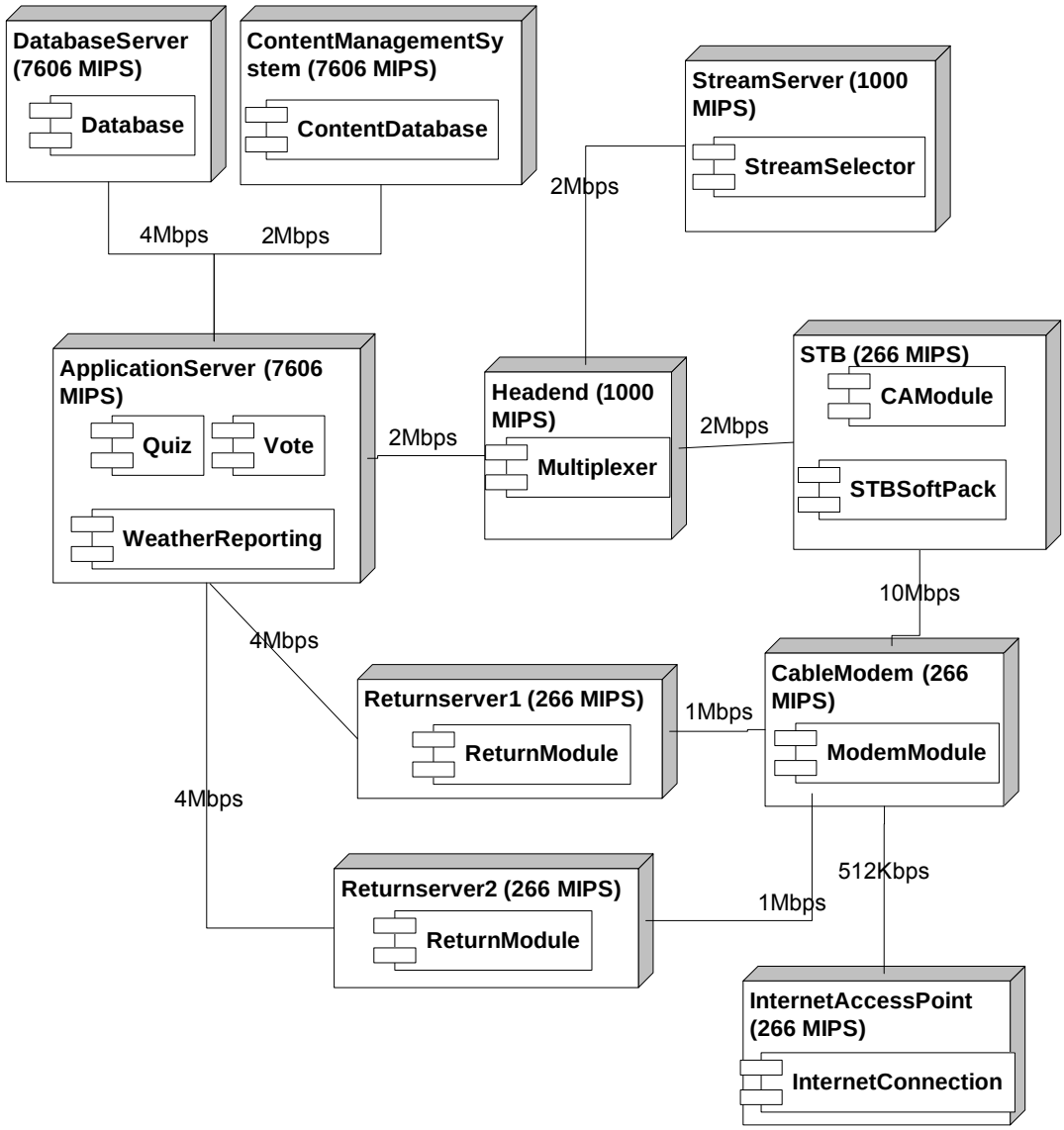


Figure 30: Alternative System Architecture

5.3.1 Reliability Influence

Reliability is improved for multiple return servers [29]. Whenever a server goes down or gets too much loads, the system administrator changes the list of available servers and broadcasts it to all STBs. This solution is credible because the broadcast channel operates independently of the cable modems. The STB deals with a change in the availability list by reconnecting to another return server when the current one is not available. Cable modem will select the proper return server to proceed with the iTV task. Therefore the parameter of return server reliability rn_i in equation (2) (see section 4.1) equals (1 - (1- reliability (return server1)) * (1- reliability (return server2))). The hot standby strategy will obviously increase the overall reliability.

In the alternative architecture, database runs on the separate database server, while it is deployed on application server in original architecture. Only application server can access the database server. The problem of original system is that the application server supports many other complex services and software components. It has much higher probability of crashing than the database server. Another problem is that application server is accessible from each STB. The database deployed on application server will have higher risk of being attacked. Therefore a separate and safe database server will decrease the failure probability of database.

5.3.2 Performance Influence

Performance is improved using the alternative architecture. At some point in the return path, there is a bandwidth limit of communication [29]. This limit creates a problem for simultaneous communication from all STB to the return servers. At the start of the iTV program, the STB not only decides randomly which return server it is connected to, but also distributes the workload.

The separate database server will also reduce the workload and disk usage on the application server, which provides real-time service to customers.

5.3.3 Cost Influence

For adding new database server and return servers, we definitely need to devote more money in building the alternative system.

5.4 Quality Models

In this section we represent the reliability model, performance model and cost model of iTV case study. The data may not be very accurate but reasonable enough to assess our case's quality attributes. The multi-objective design space analysis will be carried out against these two architectures.

5.4.1 Reliability model

Reliability data is stored into a separate XML file. It contains the reliability of each operation, arrival rate of use case, probability distribution of each scenario in a use case and reliabilities of physical hardware (i.e. nodes and node links).

	operationName	fromInterface	fromServiceName	operationReliability
1	onlineQuiz	IInteractivity	STBSoftPack	operationReliability value 0.999999
2	quizQA	ICAModule	CAModule	operationReliability value 0.999999
3	getQuizAccessPermit	ISTBCom	STBSoftPack	operationReliability value 0.999999
4	sendQuizRequest	IComm	ModemModule	operationReliability value 0.99999
5	receiveQuizData	ITransferData	ReturnModule	operationReliability value 0.99999
6	Quiz	IQuizApplication	Quiz	operationReliability value 0.999999
7	writeQuizData	IDatabase	Database	operationReliability value 0.999999
8	quizXmlGenerate	IQuizApplication	Quiz	operationReliability value 0.999999
9	QuizStat	IStastics	ContentDatabase	operationReliability value 0.999999
10	receiveQuizResults	IQuizApplication	Quiz	operationReliability value 0.99999
11	streamMultiplex	IMultiplex	Multiplexer	operationReliability value 0.99999999
12	receiveSignal	ITVShow	STBSoftPack	operationReliability value 0.99999999
13	onlineVote	IInteractivity	STBSoftPack	operationReliability value 0.999999
14	voteCA	ICAModule	CAModule	operationReliability value 0.999999
15	getVoteAccessPermit	ISTBCom	STBSoftPack	operationReliability value 0.999999

16	sendVoteRequest	IComm	ModemModule	**operationReliability** = **value** 0.999999
17	receiveVoteData	ITransferData	ReturnModule	**operationReliability** = **value** 0.9999
18	Vote	IVoteApplication	Vote	**operationReliability** = **value** 0.999999
19	writeVoteData	IDatabase	Database	**operationReliability** = **value** 0.999999
20	voteXmlGenerate	IVoteApplication	Vote	**operationReliability** = **value** 0.999999
21	VoteStat	IStastics	ContentDatabase	**operationReliability** = **value** 0.999999
22	receiveVoteResults	IVoteApplication	Vote	**operationReliability** = **value** 0.999999
23	weatherReporting	IInfoService	STBSoftPack	**operationReliability** = **value** 0.999999
24	weatherCA	ICAModule	CAModule	**operationReliability** = **value** 0.999999
25	getWeatherAccessPermit	ISTBCom	STBSoftPack	**operationReliability** = **value** 0.999999
26	sendWeatherRequest	IComm	ModemModule	**operationReliability** = **value** 0.999999
27	receiveWeatherData	ITransferData	ReturnModule	**operationReliability** = **value** 0.9999
28	weatherReport	IWeatherApplicatio n	WeatherReporting	**operationReliability** = **value** 0.999999
29	WeatherRequest	IWeather	ContentDatabase	**operationReliability** = **value** 0.999999
30	receiveWeatherResults	IWeatherApplicatio n	WeatherReporting	**operationReliability** = **value** 0.999999
31	streamSelection	IStream	StreamSelector	**operationReliability** = **value** 0.999999999
32	InternetBrowser	IInternetService	STBSoftPack	**operationReliability** = **value** 0.999999
33	httpCA	ICAModule	CAModule	**operationReliability** = **value** 0.999999
34	getHttpAccessPermit	ISTBCom	STBSoftPack	**operationReliability** = **value** 0.999999
35	sendHttpRequest	IComm	ModemModule	**operationReliability** = **value** 0.999999
36	httpRequest	IInternetAccess	InternetConnection	**operationReliability** = **value** 0.999999
37	httpResult	IComm	ModemModule	**operationReliability** = **value** 0.999999

38	showHttpResult	IInternetService	STBSoftPack	**operationReliability** = **value** 0.999999
39	SendEmail	IInternetService	STBSoftPack	**operationReliability** = **value** 0.999999
40	smtpCA	ICAModule	CAModule	**operationReliability** = **value** 0.999999
41	getSmtpAccessPermit	ISTBCom	STBSoftPack	**operationReliability** = **value** 0.999999
42	sendSmtpRequest	IComm	ModemModule	**operationReliability** = **value** 0.999999
43	SendEmailData	IInternetAccess	InternetConnection	**operationReliability** = **value** 0.999999
44	ReceiveEmail	IInternetService	STBSoftPack	**operationReliability** = **value** 0.999999
45	pop3CA	ICAModule	CAModule	**operationReliability** = **value** 0.999999
46	getPop3AccessPermit	ISTBCom	STBSoftPack	**operationReliability** = **value** 0.999999
47	sendPop3Request	IComm	ModemModule	**operationReliability** = **value** 0.999999
48	CheckMailBox	IInternetAccess	InternetConnection	**operationReliability** = **value** 0.999999
49	emailRetrieve	IComm	ModemModule	**operationReliability** = **value** 0.999999
50	showEmailReceived	IInternetService	STBSoftPack	**operationReliability** = **value** 0.999999

Reliability of Hardware Nodes		
	Nodes	Reliability
1	Stream Server	0.999999
2	Head end	0.999999
3	STB	0.999999
4	Cable Modem	0.99999
5	Internet Access Point	0.99999
6	Returnserver1	0.9999
7	Returnserver2	0.9999
8	Application Server	0.999999
9	Content Management System	0.99999
10	Database Server	0.9999999

Reliability of Connectors			
	Node End1	Node End2	Reliability
1	Stream Server	Head end	0.99999999
2	STB	Head end	0.99999999
3	Application Server	Head end	0.999999
4	Application Server	Return Server1	0.99999
5	Application Server	Content Management System	0.999999
6	Return Server1	Cable Modem	0.99999
7	Cable Modem	Internet Access Point	0.999999
8	Cable Modem	STB	0.999999
9	Application Server	Database Server	0.9999999
10	Application Server	Return Server2	0.99999
11	Cable Modem	Return Server2	0.99999

	Use case	Arrival Rate of Use case	Tasks	Probability
1	Interactivity	3	Online Quiz	0.8
			Online Vote	0.2
2	Info Service	10	Weather Report	1.0
3	TV Show	1000	TV Show	1.0
4	Internet Service	100	Internet Browser	0.8
			Send email	0.05
			Receive email	0.15

5.4.2 Cost model

Cost model contains the price information of hardware and software elements in the system. The software component costs were specified according to its source code complexity or a third-party component producer. The hardware costs were referred to the market prices. The values may not be accurate, but enough for the design comparison purpose.

Cost of Software Components		
	Components	Cost (Euro)
1	Stream Selector	2000
2	Multiplexer	2000
3	CAModule	2
4	STB SoftPack	15
5	Modem Module	20
6	Internet Connection	10
7	Return Module	5000
8	Quiz	10000
9	Vote	10000
10	Weather Reporting	5000
11	Database	4000
12	Content Database	50000

Cost of Hardware Nodes		
	Nodes	Cost (Euro)
1	Stream Server	1000
2	Head end	5500
3	STB	100
4	Cable Modem	80
5	Internet Access Point	50
6	Returnserver1	1000
7	Returnserver2	1000
8	Application Server	2000
9	Content Management System	2000
10	Database Server	2000

Cost of Connectors			
	Node End1	Node End2	Cost (Euro)
1	Stream Server	Head end	20
2	STB	Head end	10
3	Application Server	Head end	50
4	Application Server	Return Server1	100
5	Application Server	Content Management System	20
6	Return Server1	Cable Modem	10
7	Cable Modem	Internet Access Point	10
8	Cable Modem	STB	5
9	Application Server	Database Server	20
10	Application Server	Return Server2	100
11	Cable Modem	Return Server2	10

5.4.3 Performance model

Our performance model specifies the execution time of each operation called in scenarios on system architecture. The given performance requirements of system are portrayed in Table 1.

Req. ID	Scenario	Performance requirements
R1	Online Quiz	response time of the scenario to the user is less than 3000 ms
R2	Online Vote	response time of the scenario to the user is less than 3000 ms
R3	weather report	response time of the scenario to the user is less than 1000 ms
R4	TV Show	response time of the scenario to the user is less than 200 ms
R5	Internet Browser	response time of the scenario to the user is less than 2000 ms
R6	Send email	response time of the scenario to the user is less than 3000 ms
R7	Receive email	response time of the scenario to the user is less than 3000 ms

Table 1: Performance requirement of Scenarios

For computation of the workload effect on performance, we assume that the database component of original architecture takes up 60% CPU usage of application server. Another assumption is that each STB takes up 10% bandwidth of return channel in the original architecture. The bandwidth occupation of each STB rises to 20% in the alternative architecture

which offers two return channels. Also return components are executed in the alternative system twice as quick as they are in original architecture, because the workload is taken by two return servers.

For example, CPUs of the application server and database server are both 7606 MIPS (AthlonXP 2500+CPU, from AMD). The transmission speed between STB and return server is 1Mbps.Therefore CPU usage by database in the original architecture is 4563 MIPS (7606 * 60%), while it is 7606MIPS (7606 * 100%) in the alternative architecture. The transmission speed between each STB and return server is 100kbps (1Mbps * 10%), while it is 200kbps (2* 1Mbps * 10%) in the alternative architecture.

5.5 Results of Quality Assessment

5.5.1 Reliability

The component reliability model contains data on a probability of failure of each implemented operation. The system reliability is calculated as a product of:

a) Cumulative hardware reliability for the simulation period

b) Cumulative reliability of the operations invoked during the simulation period.

Based on the reliability data (see section 5.4.1) and reliability calculation algorithm in chapter 4, we implement iTV case in our tool and obtain reliabilities information of each scenario, use case and the whole system.

The found reliabilities of the two architecture alternatives are compared in Table 2. Our tool does the analysis automatically. This method used has been discussed in chapter 4.

Attribute		Original architecture	Alternative architecture
Reliability of Scenario(s)	Online Quiz	98.987496%	99.98103%
	Online Vote	98.98037%	99.97383%
Reliability of Use case "Interactivity" in one hour		96.99%	99.94%
Reliability of Scenario(s)	weather report	98.98255%	99.97417%
Reliability of Use case "Information Service" in one hour		90.28%	99.74%
Reliability of Scenario(s)	TV Show	99.999696%	99.999696%
Reliability of Use case "TV Show" in one hour		99.7%	99.7%

Reliability of Scenario(s)	Internet Browser	99.995476%	99.995476%
	Send email	99.996984%	99.996984%
	Receive email	99.995476%	99.995476%
Reliability of Use case "Internet Service" in one hour		99.56%	99.56%
Reliability of overall system in one hour		86.91%	98.94%
Failure Probability of overall system (100% - Reliability) in one hour		13.09%	1.06%

Table 2 Calculated reliability values for each architecture

From the Table 2, we see that reliabilities of certain scenarios are greatly improved whose executing paths contain return server or database server. Therefore reliability of overall system gets promotion and fits higher reliability requirement.

5.5.2 System Cost

The system cost is the sum of hardware costs and software costs. The calculated costs of architecture alternatives are given in Table 3.

Attribute	Original Arch	Alternative Arch
Cost, thousands euro	100.002	102.022

Table 3 Calculated cost values for each architecture

5.5.3 Performance

The scenario simulation results in execution time consumption of a system for each scenario. Table 4 portrays the predicted task execution time against the real-time requirements for each scenario.

Req.	Scenario	Requirement	Original Arch	Alternative Arch
R1	Online Quiz	2000ms	490ms	446 ms
R2	Online Vote	2000ms	460ms	383 ms
R3	Weather report	1000ms	295ms	269 ms
R4	TV Show	200ms	97.5ms	97.5 ms
R5	Internet Browser	2000ms	441.5ms	441.5ms
R6	Send email	2000ms	473ms	473ms
R7	Receive email	2000ms	443ms	443ms

Table 4: real-time requirements against the predicted execution time for two architectures

Performances of certain scenarios are greatly improved which pass through return server or database server.

5.6 Analysis of Architecture alternatives

The performance, reliability and cost attributes were selected as main objectives for the quality exploration. We try to find a dominant architecture solution, or a number of optimal non-dominant solutions. Pareto analysis for multi-objective optimization is a powerful means for solving conflicting objectives [32]. We obtained several two-dimensional Pareto graphs. One of them, reliability vs. cost is depicted in Figure31.

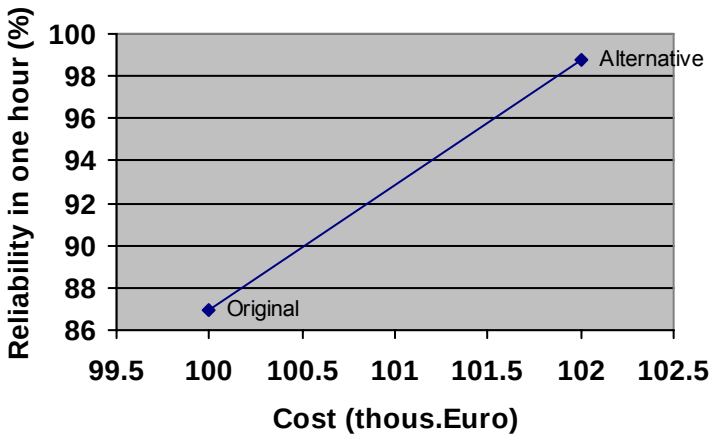


Figure31 cost-reliability exploration of the two architecture alternatives

There is no unique solution to the multi-objective optimization problem. With respect to the cost-reliability trade-off, if the cost has larger weight than reliability – the original architecture can be adopted for further development. In our case, alternative architecture will be considered as preferable architectures.

6. Implementation of the Tool

The complexity of the quality models in large-scale projects exceeds human capabilities. Therefore helpful and efficient tool support is necessary. We developed a tool to implement quality analysis.

In our project we have extended the RTIE tool of Egor Bondarev. Here we will give an outline of the tool's design and implementation. For a more detailed description of the tool refer to [26].

6.1 Environment

The RTIE tool's components are implemented using Java and Eclipse. Use case diagram and deployment diagram are drawn in EclipseUML environment [33]. We make component related input models in XML format manually. After given these necessary input models, the tool parses them, then simulates the scenarios and explores system quality based on the data parsed. Finally RTIE gives a summary report of reliability and cost quality attributes.

6.2 Objects

The RTIE tool's primary task is the quality analysis of architectures. On the other hand, it is also a case tool to architect a Robocop component-based system. Its functionality is spread across a collection of classes. The specific tasks of the classes will be explained here; in Figure 32 we have illustrated the classes' interaction.

ReliabilityFactory. Quality analysis process starts from create a new simulation object on the scenario diagram. When the user chooses the function of reliability simulation, ReliabilityFactory is initiated and then calls ApplicationXMLParser to parse input models.

ApplicationXMLParser. This object extracts the relevant application level quality information from the XML representation of the models, for example, which use cases and scenarios are implemented in the architecture.

ScenarioXMLParser. This object parses scenario model files.

TaskConstructor. This object compiles the set of models, reconstructs tasks in the scenario, and then gets a sequence of all involved operations in a scenario.

ReliabilitySimulator. This object actually performs the analysis of the architecture. After getting the array of operations in a scenario, the related reliability and cost information are filled in the properties of each operation for quality calculation. At the end of the process, the results of quality analysis are given with a pop-up dialogue.

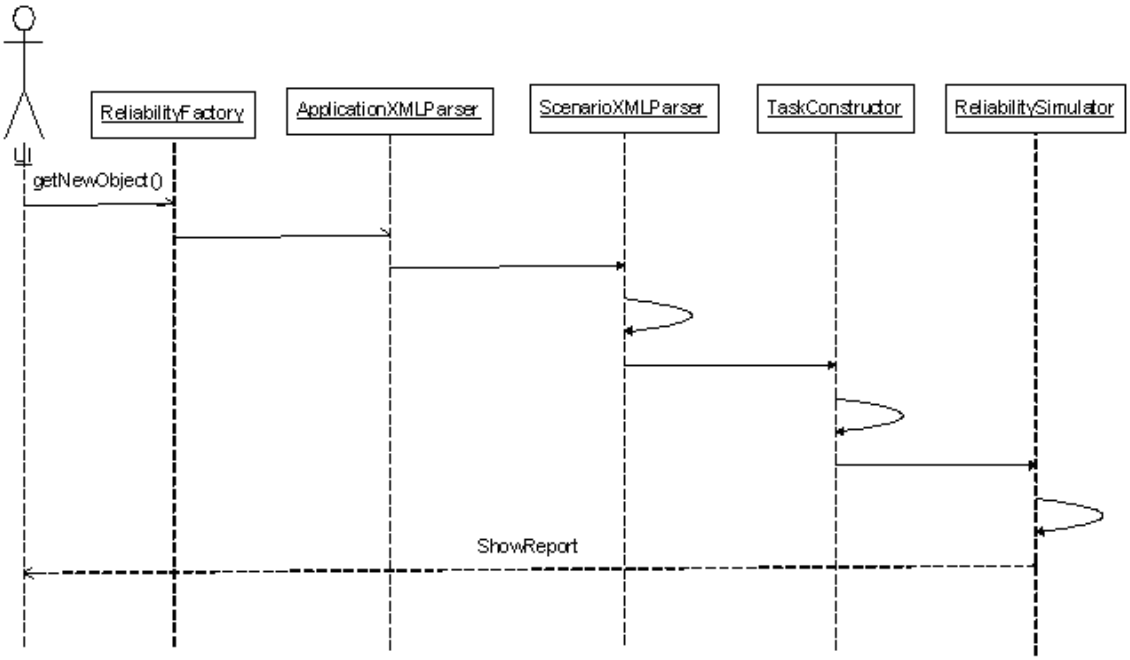


Figure 32: Message Sequence Chart of RTIE tool

6.3 Simulation and Analysis Approach

Our simulation and analysis approach is a six-step strategy. Figure 33 depicts the architecture and process of the RTIE tool.

1st step: the third-party component producers provide behavior models and quality models (e.g. reliability model, resource model etc.) of components.

2nd step: System architects compose selected components and specify scenario model for each execution scenario. Then they define hardware architecture and map the software components on it. A Robocop CBA is then accomplished.

3rd step: All these models are aggregated and construct an application scenario model.

4th step: Our tool compiles the application scenario model and reconstructs the tasks. At the end of the step, a pool of tasks in system is generated.

5th step: Our tool simulates each execution scenario and assesses these scenarios based on component quality models provided by component producers.

6th step: After accomplished the analysis approaches, tool offers a report of system quality attributes.

For details about RTIE tool's capability of predicting system performance, or realization of simulation, please refer to [26].

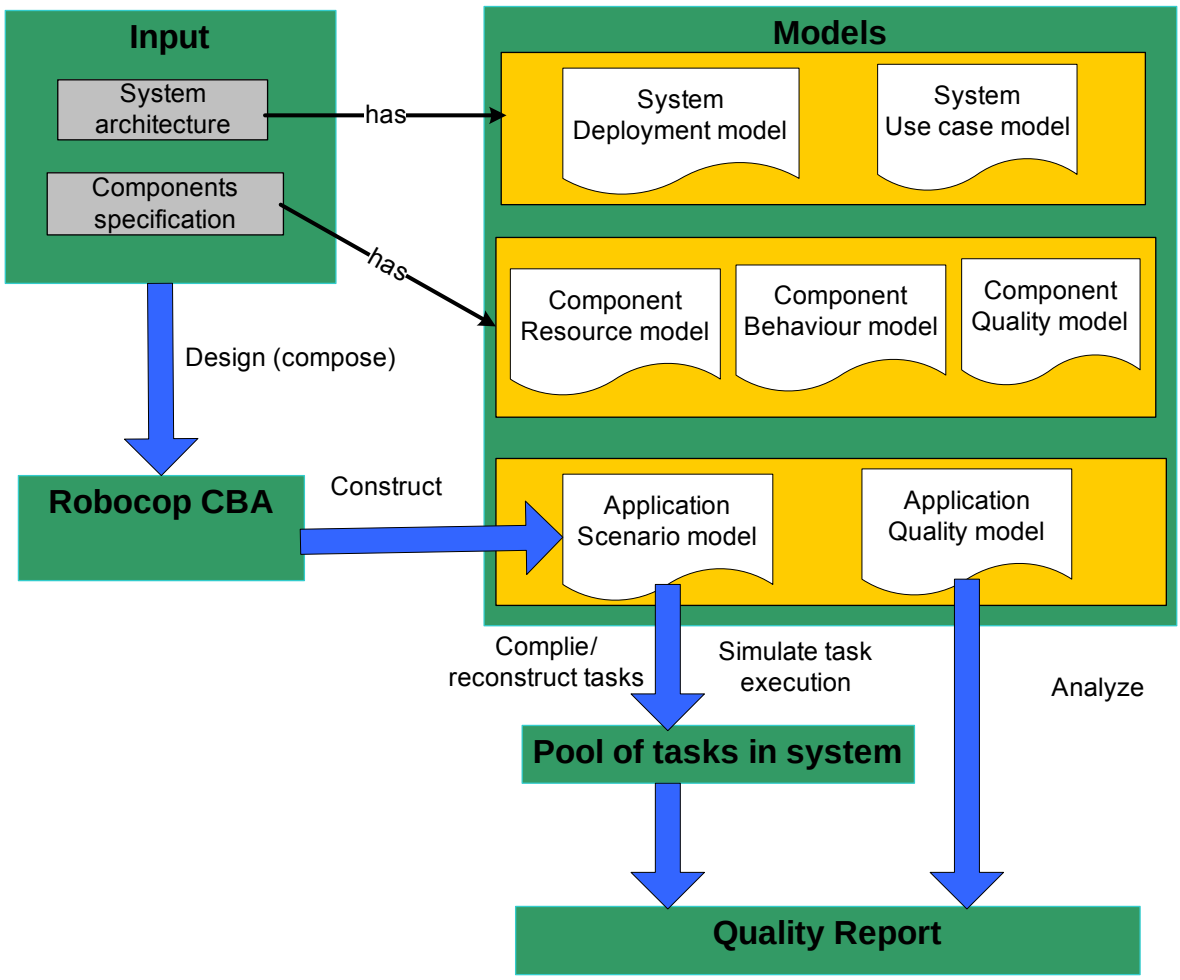


Figure 33: Architecture and process of RTIE tool

7. Conclusion

This chapter briefly summarizes the problem and what we established during this project.

My project aimed to develop quantitative techniques for evaluating quality properties of component-based architectural designs and to implement these techniques such that they can be used in the Robocop Integration Environment. Hence it is tailored to the Robocop Component architecture and its associated models. In this project, we mainly focus on techniques for reliability and cost analysis. The specification required as input to the quality calculation algorithm is compatible with the "4+1 views"-manner of describing software designs.

The scale and complexity of many projects is so large that efficient tool support is necessary. We developed a tool to implement quality analysis. This tool calculates the relevant quality properties for a given architectural alternative. We assume that the reliability and cost of the components in the architecture are given. The tool is then able to predict the relevant quality properties of the overall system.

Two case studies were conducted to validate the proposed techniques and to assess the reliability characteristics of industrial models. One of them is the car navigation system; the other is the Interactive-TV (iTV) system. The results obtained can be used for selecting the best solution from alternative architectures. Also, these analysis help in detecting possible quality bottlenecks of the system, or in choosing which software component could promote system's quality most.

Calculating reliability using our method requires a large number of input parameters. In practice this quality data of software and hardware components devices is often not or only partially available. Solutions to this are to work with estimates or with probability distributions.

8. Future work

In this project we encountered the problem that in practice the quality models are often incomplete. Complete quality information increases the accuracy of our prediction and enables avoiding unqualified system due to wrong selection of third-party components.

In the aspect of theory, it is very interesting to research on prediction technology in the case of incomplete inputs. A possible approach is to construct a probability distribution, which enables the estimation of a system's quality based on incomplete information.

In the future we need user interface toolkit for inputting quality data and improving interaction between machine and users. That would also be useful for maintaining consistency between models.

Bibliography

[1] Egro Bondarev, Michel R.V. Chaudron, Jiang Zhang and Arjen Klomp. Quality-Oriented Design Space Exploration for Component-Based Architectures, Technical Report, Technical University of Eindhoven, 2005

[2] K.Deb. Multi-objective optimization using evolutionary algorithms. John Wiley, Chichester, 2001

[3] P.Lieverse, P.van der Volf, K.Vissers, "A Methodology for Architecture Exploration of Heterogeneous Signal Processing Systems", Journ. VLSI Signal Proc. Signal, Image and Video Proc, vol. 29, pp. 197-207, 2001

[4] L.Thiele, S.Chakraborty, M.Gries, and S.Kunzali. "A framework for evaluating design tradeoffs in packet processing architectures", In Proc. 39th DAC conference, New Orleans, USA, 2002. ACM Press.

[5] K.Lahiri, A.Raghunathan, and S.Dey. "System level performance analysis for designing on-chip communication architectures". IEEE Trans. On Computer Aided-Design of Integrated Circuits and Systems, 20(6):768783, 2001

[6] M. Grunewald, et al. "A framework for design space exploration of resource efficient network processing on multiprocessor SoCs". In Procs. 3rd Workshop on Network Processors and Applications, Madrid, Spain, February, 2004.

[7] M.Vachharajani. Microarchitecture "Modeling for Design-space Exploration". PhD thesis, Princeton University, November, 2004.

[8] J. T. Russell, "Architecture-level performance evaluation of component-based embedded systems", In Proc. 40th DAC conference, Anaheim, USA , 2003. ACM Press.

[9] S. Mohanty et al., "Rapid Design Space Exploration of Heterogeneous Embedded Systems using Symbolic Search and Multi-Granular Simulation", In Procs. LCTES02-SCOPES02, June, 2002, Berlin, Germany.

[10] S. M. Yacoub et al., "Scenario-Based Reliability Analysis of Component-Based Software", In Procs. 10th Int. Symp. on Software Reliability Engineering November, 1999, Boca Raton, Florida.

[11] S. Krishnamurthy, A.P. Mathur, "On The Estimation Of Reliability Of A Software System Using Reliabilities Of Its Components", In Procs. ISSRE 97 November 1997, Albuquerque, USA.

[12] R. Roshandel et al. "Toward Architecture-Based Reliability Estimation", In Procs. WADS conference, Edinburg, UK, 2004.

[13] Len Bass, Paul Clements, Rick Kazman. "Software Architecture in Practice(2nd Edition)". Addison Wesley, 2003 ISBN: 0321154959

[14] Christian F.J.Lange "Empirical Investigations in Software Architecture Completeness". Master thesis, Technische Universiteit University, September, 2003.

[15] "How Do You Define Software Architecture?", Software Engineering Instituete, Carnegie Mellon University.
[http://www.sei.cmu.edu/architecture/definitions.html]

[16] Booch, Rumbaugh, and Jacobson, "The UML Modeling Language User Guide", Addison-Wesley, 1999

[17] M. Shaw, D. Garlan "Software architecture: perspectives on an emerging discipline". Prentice Hall, New Jersey, 1996. ISBN: 0131829572

[18] Philippe Kruchten. Architectural Blueprints – "The 4+1 View Model of Software Architecture", IEEE Software 12(6), pp 42-50. November 1995

[19] Robocop public homepage.
[http://www.extra.research.philips.com/euprojects/Robocop/]

[20] R. van Ommering et al., "The Koala component model for consumer electronics software", IEEE Trans. Computer, 33(3), 78-85, Mar. 2002

[21] IEEE Std 610.12-1990. Glossary of Software Engineering Terminology. Institute of Electrical and Electronics Engineers, Inc., 1990.

[22] ISO 9003-3-1991. Quality Management and Quality Assurance Standards, Part 3: Guidelines for the Application of ISO 9001 to the Development, Supply and Maintenance of Software. International Standards Organization, 1991.

[23] Schulmeyer, G. Gordon, and James I. McManus. Handbook of Software Quality Assurance, 3rd ed. Prentice Hall PRT, 1998

[24] American Institute of Aeronautics and Astronautics, *Recommended Practice for Software Reliability*, ANSI/AIAA R-013-1992, February 1993.

[25] E.Wandeler, L.Thiele, M.Verhoef, "System Architecture Evaluation Using Modular Performance Analysis – A Case Study". 1st International Symposium on Leveraging Applications of Formal Method (ISoLA), Paphos, Cyprus. 2002.

[26] E.Bondarev, J.Muskens, P.H.N de With and M.R.V. Chaudron, "Predicting Real-Time Properties of Component Assemblies: a Scenario-Simulation Approach", Proc. 30th Euromicro Conf., CBSE Track, ISBN 0-7695-2199-1, pp. 40-47, September 2004.

[27] Vittorio Cortellessa, Harshinder Singh, Bojan Cukic, "Early reliability assessment of UML based software models". 3rd ACM Workshop on Software and Performance, July 24-26, 2002 – Rome (Italy).

[28] Altova XMLSpy® 2005 , http://www.altova.com/products_ide.html

[29] Jan L.M. Verhoeven, Peter H.N. de With, Wim J.C.M. Bus, "System architecture for Experimental Interactive Television". Proceedings of the 3rd PROGRESS Workshop on Embedded Systems, 24 October 2002, Utrecht, Netherlands.

[30] Interactive TV Web, http://www.interactivetvweb.org/

[31] COTS-Based Systems(CBS) Initiative, http://www.sei.cmu.edu/cbs/

[32] Mattson C.A., and Messac A., "A Non-Deterministic Approach to Concept Selection using s-pareto Fontiers", Proceedings of ASME DETC 2002, Montreal, Canada, September 2002

[33] EclipseUML website, http://www.omondo.com/

[34] Ralf H. Reussner , Heinz W. Schmidt , Iman H. Poernomo, Reliability prediction for component-based software architectures, Journal of Systems and Software, v.66 n.3, p.241-252, 15 June 2003

[35] A. Abd-allah. Extending reliability block diagrams to software architectures. Technical report, Center for Software Engineering, Computer Science Department, University of Southern California, Los Angeles, CA 90089 USA, 1997.

[36] James A. Whittaker , Michael G. Thomason, A Markov Chain Model for Statistical Software Testing, IEEE Transactions on Software Engineering, v.20 n.10, p.812-824, October 1994

[37] H.Schmidt and R.H.Reussner, Parametrized Comtracts and Adapter Synthesis, In Proceedings of 5th ICSE workshop on CBSE, 2001