Adaptability Evaluation at Software Architecture Level

Pentti Tarvainen*

VTT Technical Research Centre of Finland, Kaitoväylä 1, P.O. Box 1100, FIN-90571 Oulu, Finland

Abstract: Quality of software is one of the major issues in software intensive systems and it is important to analyze it as early as possible. An increasingly important quality attribute of complex software systems is adaptability. Software architecture for adaptive software systems should be flexible enough to allow components to change their behaviors depending upon the environmental and stakeholders' changes and goals of the system. Evaluating adaptability at software architecture level to identify the weaknesses of the architecture and further to improve adaptability of the architecture are very important tasks for software architects today. Our contribution is an Adaptability Evaluation Method (AEM) that defines, before system implementation, how adaptability requirements can be negotiated and mapped to the architecture, how they can be represented in architectural models, and how the architecture can be evaluated and analyzed in order to validate whether or not the requirements are met. AEM fills the gap from requirements engineering to evaluation and provides an approach for adaptability evaluation at the software architecture level. In this paper AEM is described and validated with a real-world wireless environment control system. Furthermore, adaptability aspects, role of quality attributes, and diversity of adaptability definitions at software architecture level are discussed.

Keywords: Adaptability, adaptation, adaptive software architecture, software quality, software quality attribute.

INTRODUCTION

Today, quality of a software system plays an increasingly important role in the domain of software engineering. It is commonly accepted that crucial quality attributes, such as performance, scalability, reliability, availability, integrability, extensibility, and maintainability, as well as adaptability are heavily influenced by the software architecture. A systematic understanding of how architectural design decisions affect the system's quality is lacking. To treat software design as an engineering discipline rather than an art, we need the ability to address the quality of the software architecture directly on the architecture model level, not simply as it is reflected in the implemented system. Quality requirements for the final software system can be determined at the software architecture level by means of the quality attributes [1, 2]. Quality goals, quality attributes, quality requirements or, non-functional requirements [1, 3-5] answer to the question "how well" whereas software functional requirements answer to the question "what" [6].

Software architecture is defined as "the fundamental organization and behavior of a system in terms of components and connectors" [1]. Furthermore, in literature, there has been defined at least seven different meanings for software architecture [6]. In general, architectural models document architecture to the body of knowledge for reusing the architecture at multiple levels of granularity [6-9]. Quite recently some guidelines for software architecture documentation, such as [8, 10], have emerged [6].

Architecture models are an expression of the earliest design decisions [1, 8, 9] and a means of abstraction [8, 11] to understand the system [6]. Examples of design decisions are the decisions such as "we shall separate user interface from the rest of the application to make both user interface and application itself more easily modifiable" [6]. Expressions of the design decisions are many and they may even be as small as definition of components and connectors. Furthermore, software architecture models can be seen as the language for communication [1, 6, 8, 9, 11]. The role of architecture models is also to provide analysis opportunities at early stages of development [6-8]. Architecture model is also an expression of the system's evolution [6-8] and a management instrument [6, 8, 11].

As the meanings of software architecture are many, the role of software architect has become very demanding. The level of abstraction has risen, required amount of cumulative knowledge has exploded and international and multicultural environments with geographically distributed development sites emphasize an ability to communicate ideas clearly. Clements *et al.* [12] made quite an extensive survey on the duties, skills and knowledge required from software architects today [6]. The survey covered e.g. web pages, books, job descriptions and university courses on software architecture. This study considered that software architect and quality analyst play a role and therefore, the duties of software architect include project and requirements management and also architecture evaluation and analysis duties. In addition to communication skills, an architect needs the skill for abstraction, i.e. skills for handling the unknown and skills for handling the unexpected. These are two different but related skill sets. Skills are important but, useless without competent and appropriate knowledge on e.g. computer science, architecture concepts, technologies and platforms, programming and knowledge on organization's context and management.

Technology, environment, and user requirements have changed rapidly in the domain of software engineering. Con-

*Address correspondence to this author at the VTT Technical Research Centre of Finland, Kaitoväylä 1, P.O. Box 1100, FIN-90571 Oulu, Finland; E-mail: pentti.tarvainen@vtt.fi sequently, adaptability has become one of the key feature and an important factor for survival and success of software systems and it attracts the concentration of both; researchers and industry. Most essential aspects of the software systems adaptation are their (1) ability to observe their runtime behavior and interpret those observations in terms that permit a high-level understanding of their status, (2) capability to adapt in order to accommodate variable resources, system errors, and stakeholders' changing requirements, (3) capability to adapt their functionality, even at runtime, to behavioral and structural changes that occur either internally or externally in their operating environment and without any external human intervention, and (4) ability to allow components to change their pattern depending upon the environmental changes and goals of the software system, without changing the actual components themselves. In this paper, adaptability aspects, role of the quality attributes, and diversity of adaptability definitions at software architecture level are discussed. Because there is no explicit, extensive and concrete definition related to the software system and the software architecture level adaptability, a new definition for that is discussed and proposed.

Software architectures for complex software systems should be flexible and reflective enough to allow components to change their pattern. These environmental changes should be realized according to the evolving context and execution environment so that software systems can stay compliant with the specifications and requirements of the software systems. Analyzing the software adaptability at the architecture level to identify the weakness of the architecture and further to improve adaptability of the architecture are very important issues for software professionals today. All of adaptability requirement sources should be identified and the requirements should be negotiated in a way that the best possible requirement set can be identified. The problem in adaptability of today's software systems should be able to be analyzed before system implementation, i.e. when the corrections and modifications are easier and cheaper to perform and the design decisions can still be affected.

Our contribution is Adaptability Evaluation Method (AEM) [13, 14] that defines (1) how adaptability requirements can be negotiated and mapped to the architecture, (2) how adaptability can be represented in the architecture models, and (3) how the architecture can be analyzed in order to validate whether or not adaptability requirements are met. AEM fills the gap between requirements engineering and evaluation and provides a systematic framework and notation extensions, techniques and guidelines for adaptability evaluation at the software architecture level. In this paper, AEM is described and validated with a real-world wireless environment control system. AEM is an integral part of QADA®[1] (Quality-driven Architecture Design and quality Analysis) methodology [15] focusing its activities in adaptability-related aspects. QADA® also provides other evaluation methods, e.g. RAP (Reliability and Availability Prediction) [16] and IEE (Extensibility and Integrability Evaluation) [6], for other quality attributes.

Quality-driven software architecture development emphasizes the importance of quality attributes, wherein quality attributes refer to the non-functional properties of software products. The approach relies on gathering, categorizing and documenting quality properties as at least equally important requirements as functional requirements and constraints, and utilizing the gained knowledge in architectural design. The quality-driven design is further complemented with an architectural analysis. Architectural analysis is about testing the architecture model produced in the design, i.e. verifying whether the architecture meets the quality goals set in the very beginning. These two activities combined together form an interacting pair of activities in software architecture development.

The structure of the paper is as follows. The next section discusses briefly about related work. After that, adaptability aspects and dimensions, and the role of the quality attributes related to adaptability at software architecture level are discussed. In addition, a definition for the system and the software architecture level adaptability is discussed. The next section describes the target system and the objectives of the case and after that an introduction to AEM is provided. After that, the phases and steps of AEM, and how they can be applied to the case, are described and discussed. Discussion summarizes our experiences on using the method and concludes the paper. Finally, in the last section, ideas for future work are discussed.

RELATED WORK

There are various quality evaluation methods and techniques available, e.g. for evaluating interoperability [17, 18], extensibility [19], architecture mismatches [18, 20] and multiple quality attributes [21]. Based on our knowledge there is no method for adaptability evaluation that would cover software development from adaptability requirements specification to architecture design and that would enable quality evaluation from architecture models. Scenario development and scenario evaluation are the common activities for all scenario-based methods. The main differences between methods are; how early in the software architecture design the method is used, what quality attributes the method supports and how easy it is to apply and integrate to the design process [22].

Literature contains many stand-alone techniques proposed to deal with the system or architecture level adaptability, but it only contains a few evaluation or analysis methods related to the software architecture adaptability. Methodologies like ATAM [23], ALRRA [24] and ALMA [25] are focusing on the quality attributes of software systems like performance, modifiability, flexibility, maintainability, portability, variability and trade-offs between them, but none of them focuses directly on adaptability characteristic of software architectures. In paper [26] a descriptive method for analyzing self-adaptive software is proposed. The method proposes that self-adaptive software should include at least two components: (1) the deliberative component and (2) the reactive component. The method is based on feedback control and feed forward control theory. This method does not give the definition for adaptability and does not analyze adaptability deeply. The method is only a coarse qualitative analysis method. In paper [27] a process oriented metric for software architecture adaptability is presented. The method analyses the degree of adaptability through the intuitive de-

[1] QADA® is registered trademark of VTT, Technical Research Centre of Finland

composition of goals and the intuitive scoring for the goal satisfying level of software architecture. The method can find some defaults in the architecture, but it depends too much on the intuition and the expert expertise, which leads to much uncertainty. Paper [28] proposes a quantitative evaluation approach based on adaptability scenario profile, impact analysis on them and calculation of adaptability degree of them. However, the method does not consider evaluation of qualitative aspects of the software architectures.

ADAPTABILITY RELATED TO SOFTWARE ENGINEERING

Because of rapid changes of environment and requirements in software engineering, adaptability has become one of the key features of software systems and attracts the concentration of both; researchers and industry. In this section, adaptability aspects, diversity of adaptability definitions, and role of quality attributes are discussed. Because there is no explicit, extensive, and concrete definition for the software system adaptability or the software architecture adaptability, a new definition for that is discussed and proposed.

Adaptability Aspects

In literature, adaptability, related to software engineering, is considered with the terms of (1) robustness of software, (2) internal and external adaptation, (3) runtime and dynamic adaptability, (4) resource variability, (5) dynamic reconfiguration, (6) reflective adaptation and (7) self-management, which all have a particular impact on system, architecture,

Table 1. Adaptability Aspects

Adaptability Aspect	Description	Ref.
Robustness of software	Ability of a software system to tolerate some deviations in the environment	[29]
Internal adaptability	Ability of a software system to use generic mechanisms such as exception handling, or heartbeat mechanisms to trigger application-specific responses to an observed fault	[17]
External adaptability	Ability of a software system to manage adaptation outside of the system by monitoring various systems attributes, such as resource utilization, reliability, and delivered quality of service. Monitored information and external mechanisms decide whether the application must be reconfigured	[17]
Runtime adaptability:	Ability of a software system to adapt itself to changes that occur either internally or externally in its operating environment.	[30]
• Behavioral adaptability	System structure remains the same while the architectural elements of the system can be modified or replaced.	[30]
• Structural adaptability	System behavior remains the same while the configuration of the architectural elements changes.	[30]
Resource variability	Ability of a software system to manage resource variability, changing user needs, and system faults.	[31]
Dynamic adaptability	Ability of a software system to adapt to behavioral and structural changes that occur either internally or externally in their operating environment while the system is running and without bringing it down. Takes into account the evolution of the execution context and environment alternations.	[32]
Dynamic reconfiguration	Takes place during maintenance or when a new version of the system is installed. Dynamic reconfiguration has to address the issues of structural changes (topology of the application), geographical changes (distribution of the components and their localization), interface modification (changing the interface of a component), and implementation modification (changing the internals of a component, modifying its execution or updating its execution schema).	[31]
Reflective adaptation	Ability of a software system to exploit computational reflection, which allows a system to observe and modify the properties of its own behavior, especially those ones that can be observed externally, Is a solution to create applications capable to maintain, use or change the representation of their own designs.	[33]
Self-management:	Ability of a software system to be efficient without user intervention.	[34]
• Self-adapting	Ability of a software system to modify its own behavior in response to changes in its operating environment in order to provide and improve functionalities and performances.	[35]
• Self-organizing	Ability of components of software system to configure autonomously interactions with other components, guaranteeing system constraints.	[36]
• Self-configuring	Ability of a software system to automatically adapt to dynamic changes (i.e. insertion, replace, and removal of a component) of the system.	[37]
• Self-protecting	Ability of a software system to observe external world and notice and react to possible attacks to the system.	[34]
• Self-healing	Ability of a software system to examine, detect, diagnose, react and take corrective actions to the software and hardware malfunctions automatically.	[32]
• Self-optimizing	Ability of a software system to monitor and autonomously optimize system resources. Components and systems continually seek opportunities to improve their own performance and efficiency.	[38]

and component dimensions. Table **1** summarizes adaptability aspects with their descriptions and example references.

Role of Quality Attributes

Quality of software is one of the major issues in software intensive systems and it is important to analyze it as early as possible. Analyzing of software quality can be done from the descriptions of software architecture by means of quality attributes [1, 2]. ISO standard 9126-1 [4] defines a Software Quality Model including six main categories of quality attributes as follows: functionality, reliability, usability, efficiency, maintainability and portability. Furthermore, each of these quality categories is divided into several sub characteristics.

System requirements are defined as the top level requirement set consisting of software, hardware and mechanical requirements [39]. This paper focuses on software requirements and ignores the other ones. Software requirements can be defined as consisting of functional requirements and non-functional requirements (NFRs), also referred to as quality attributes or system properties [39]. The functional requirements are related to the domain functionality of the application. Typically, a functional requirement is implemented by a subsystem or a group of components in the software. Quality attributes can be categorized in development quality attributes and operational quality attributes [39]. Development quality attributes are characteristics of the system that are relevant from a software engineering perspective, e.g. maintainability, reusability, and flexibility [39]. Operational quality attributes are characteristics of the system in operation, e.g. performance, reliability, robustness and fault-tolerance [39]. Different from functional requirements, non-functional requirements can generally not be pinpointed to a particular part of the application but are a property of the application as a whole.

Requirements of software architecture include functional requirements and NFRs. NFRs have a critical role in the development of a software system as they can be used as selection criteria to help designers with rational decision-making among competing designs, which in turn affects a system's implementation. The problem of effectively designing and analyze software architecture to meet its NFRs is critical to the system's success. The significant benefits of such work include detecting and removing defects earlier, reducing development time and cost while improving the quality. The inherent nature of the NFRs makes their common understanding difficult [29]. The problem is compounded by the fact that the requirements for any software architecture are usually vague about the NFRs that the architecture should satisfy and how to analyze the NFRs in the final architecture. While there are several NFRs such as performance, maintainability, reusability, security, and so on, among the more important of the NFRs is adaptability [40].

Despite the fact that the categorization into functional and non-functional quality attributes are widely approved and used in software development, other categorizations are represented too [2]. This kind of a categorization is described e.g. in [5] defining a framework for representing the design process of non-functional requirements (or quality attributes). However, the framework does not categorize the quality attributes explicitly but concentrates on recording the reasoning process behind the design decisions [2].

With an interpretation discussed so far, the definitions of quality attributes and categorization of them to operational and development quality attributes can be defined as shown in Table **2** and Table **3**.

Table 2. Operational Quality Attributes [2]

Quality Attribute	**Description**
Performance	Responsiveness of the system, which means the time required to respond to stimuli (events) or the number of events processed in some interval of time.
Security	The system's ability to resist unauthorized attempts at usage and denial of service while still providing its service to legitimate users.
Availability	Availability measures the proportion of time the system is up and running.
Usability	The system's learnability, efficiency, memorability, error avoidance, error handling and satisfaction concerning the users' actions.
Scalability	The ease with which a system or component can be modified to fit the problem area.
Reliability	The ability of the system or component to keep operating over the time or to perform its required functions under stated conditions for a specified period of time.
Interoperability	The ability of a group of parts to exchange information and use the one exchanged.
Adaptability	The ability of software to adapt its functionality according to the current environment or user.

Table 3. Development Quality Attributes [2]

Quality Attribute	**Description**
Maintainability	The ease with which a software system or component can be modified or adapt to a changed environment.
Flexibility	The ease with which a system or component can be modified for use in applications or an environment other than those for which it was specifically designed.
Modifiability	The ability to make changes quickly and cost-effectively.
Extensibility	The systems ability to acquire new components.
Portability	The ability of the system to run under different computing systems: hardware, software or combination of the two.
Reusability	The system's structure or some of its components can be reused again in future applications.
Integrability	The ability to make the separately developed components of the system work correctly together.
Testability	The ease with which software can be made to demonstrate its faults.

The definitions of the quality attributes in Table **2** and Table **3** are based on descriptions discussed in [17] extended by definitions of the quality attributes named adaptability and extensibility. In IEEE Standard 610.12 [41], adaptability is defined as a synonym for flexibility and in ISO/IEC 9126-1 [4], as a sub quality attribute of portability. However, the meaning of adaptability, flexibility, portability, and extensibility is clearly different and adaptability exists even in different quality category than the others. According to [2] adaptability of software is "the ability of software to adapt its functionality according to the current environment or user", whereas the strict meaning of flexibility is about "easy adaptation of software to environments other than those for which it was specifically designed". On the other hand, portability of software system means the capability of the software product to be transferred from environment to other [4], whereas extensibility emphasizes the systems capability to acquire new components.

In Table **2**, adaptability is classified to category of operational quality attributes. On the other hand, Table **3** defines two development quality attributes; maintainability and modifiability, which both have aspects and references related to the characteristics of adaptability or adaptation. Consequently, adaptability is also present in Table **3** in the definitions of maintainability and modifiability in form of development characteristics of the quality attributes. As a conclusion, the lists of quality attributes are sensitive to changes in a similar way to attractiveness of systems' quality attributes [2], and furthermore, adaptability has both operational and development characteristics.

Although the Software Quality Model [4] includes functionality, i.e. the system's ability to do the work for which it was intended, it can be seen as a main category of operational quality attributes, realizing that functionality cannot be considered as an architectural quality attribute [1, 2]. However, adaptability, as well as interoperability and reliability, can be considered as special, newly emerged forms of functionality, the forms that are architectural in nature [2, 17]. While the characteristics of software systems are changing from monolithic to modular distributed network systems, and furthermore, to self-managing nets of adaptive computing units, adaptability, as well as interoperability and reliability, can be considered as a characterization of qualitative properties of software systems [2].

With interpretation discussed so far, and because adaptability related to the software systems has operational and development characteristics, and furthermore, qualitative properties, adaptability can be considered as a sub quality attribute of maintainability. In Table **3** maintainability is defined as "the ease with which a software system or component can be modified or adapt to a changed environment". Modifications may include extensions, porting to different computing systems or improvements. Maintainability is also affected by other development quality attributes. In Table **3** modifiability is defined as "an ability of software system to make changes quickly and cost-effectively". Modifiability includes adding, deleting and changing software structures and therefore, extensibility and portability can be considered as a special form of modifiability. In addition, flexibility, reusability, testability and integrability contribute to modifiability and therefore they can be defined as sub quality attributes of maintainability. Furthermore, adaptability as well as maintainability has an impact on three abstraction levels named system, architecture and component dimension. Adaptability concerns the whole life cycle of software system, and therefore, it exists at all abstraction levels in software development. In these dimensions adaptability means different things, and therefore, the techniques to achieve it also vary.

Definition of Adaptability

In literature, adaptability is defined variously. For example, in ISO/IEC 9126-1 [4] the software adaptability has been defined as "the capability of the software product to be adapted for different specified environments without applying actions or means other than those provided for this purpose for the software considered". In [42] adaptability is defined as "the system which can adjust its behavior according to changing of the environments". Furthermore, [2] define adaptability as "the ability of software to adapt its functionality according to the current environment or user". Although these different notions, there is no explicit, extensive, and concrete definition for the software system adaptability or the software architecture adaptability.

As a conclusion, adaptability, related to software engineering, has many facets, including characteristics from both functional and non-functional quality attributes. The latter quality attributes (i.e. operational and development quality attributes) can be seen as architectural in nature. Consequently, adaptability can be considered as a characterization of qualitative property of maintainability of software architecture and it should be taken into account at architectural design phase of the software system. Furthermore, adaptability includes runtime requirements of the software system as well as changes in requirements of stakeholders' objectives. We define the software system or the software architecture adaptability as follows:

Adaptability of software system or software architecture is (1) a qualitative property of its maintainability and (2) an ability of its components to adapt their functionality, even at runtime, to behavioral and structural changes that occur either internally or externally in their operating environment and in requirements of stakeholders' objectives.

In the definition above, adaptability is related to the objectives of stakeholders of the system and to the quality attributes of the software architecture. Typical stakeholders are customers, business managers, architects and architectural evolution strategists of organization. Stakeholders have different viewpoints and demand different adaptable content. For example, from the point of view of end-user, adaptability may mean that new functions can be added and deployed to the system. On the other hand, for the architects, adaptability may mean that the system can adapt with different operating systems.

INTRODUCTION TO AEM

AEM [13, 14] is an integral part of QADA® (Quality-driven Architecture Design and quality Analysis) methodology [15] specializing its activities in adaptability-related aspects. QADA® is a methodology that provides a set of methods and techniques to develop high-quality software architectures for single systems and system families. The focus of QADA® methodology is on identifying as many as possible of the design problems and quality goals in architecture design and analysis. In the design, this is achieved by identifying system stakeholders, analyzing target system quality goals from the point of view of several stakeholders and describing the architecture and quality with models from various viewpoints so that the appropriate knowledge reaches each stakeholder. The analysis considers quality goals of architecture and products from the point of view of at least developers, users and customers. QADA® uses quality requirements as a driving force when selecting software structures and it describes the architecture on two abstraction levels: conceptual and concrete (Fig. (**1**)).

The conceptual level means design decisions concerning, for example, functionality. The concrete level refines the conceptual designs in more detailed descriptions. The conceptual and concrete levels consist of four viewpoints: structural, behavioral, deployment and development. The structural viewpoint describes the compositional structure of the system, whereas the behavioral viewpoint concerns the behavioral aspects of the architecture. The deployment viewpoint allocates the components to various computing environments.

Finally, the development viewpoint presents the components, their relationships to each other and the actors responsible for their development.

AEM means capturing adaptability requirements of the software architecture and defines (1) how adaptability requirements can be negotiated and mapped to the architecture, (2) how adaptability can be represented in the architecture models, and (3) how the architecture can be analyzed in order to validate whether the requirements are met. AEM assists in requirement engineering, architecture modeling and adaptability evaluation in the architecture models, ensuring that the requirements are met before system implementation. AEM consists of three main phases, and they can be applied separately to individual systems. Each phase includes several steps, which, in turn, consist of a set of activities. These three main phases; (1) defining adaptability goals, (2) representing adaptability in architecture models, and (3) adaptability evaluation, as well as their steps are described more detailed in chapter “Applying the Phases to the Case”.

DESCRIPTION AND OBJECTIVES OF THE CASE

The target system of the case has been a wireless environment control system (Fig. (**2**)). The system is used to manage the electrical control appliances of doors, windows, lights, security systems, elevators, etc., located in the user's close surroundings. The target system consists of the Client Software installed in the user's mobile phone, and an unlimited number of the Receivers, including their own software and hardware. The receivers are connected to the electrical appliances with cable and receive control messages from the mobile phone *via* a Bluetooth interface.

An objective of the case has been to validate AEM with a real-world industrial case, emphasizing (1) the efficiency of the design and analysis of the software system architecture to meet its adaptability, (2) the software system's adaptability to multiple platforms, and (3) how adaptability mechanisms can be added to the architectures of software systems. The industrial partner has participated in the case with development process of the target system and their objectives have been (1) to develop a new adaptable software architecture for the

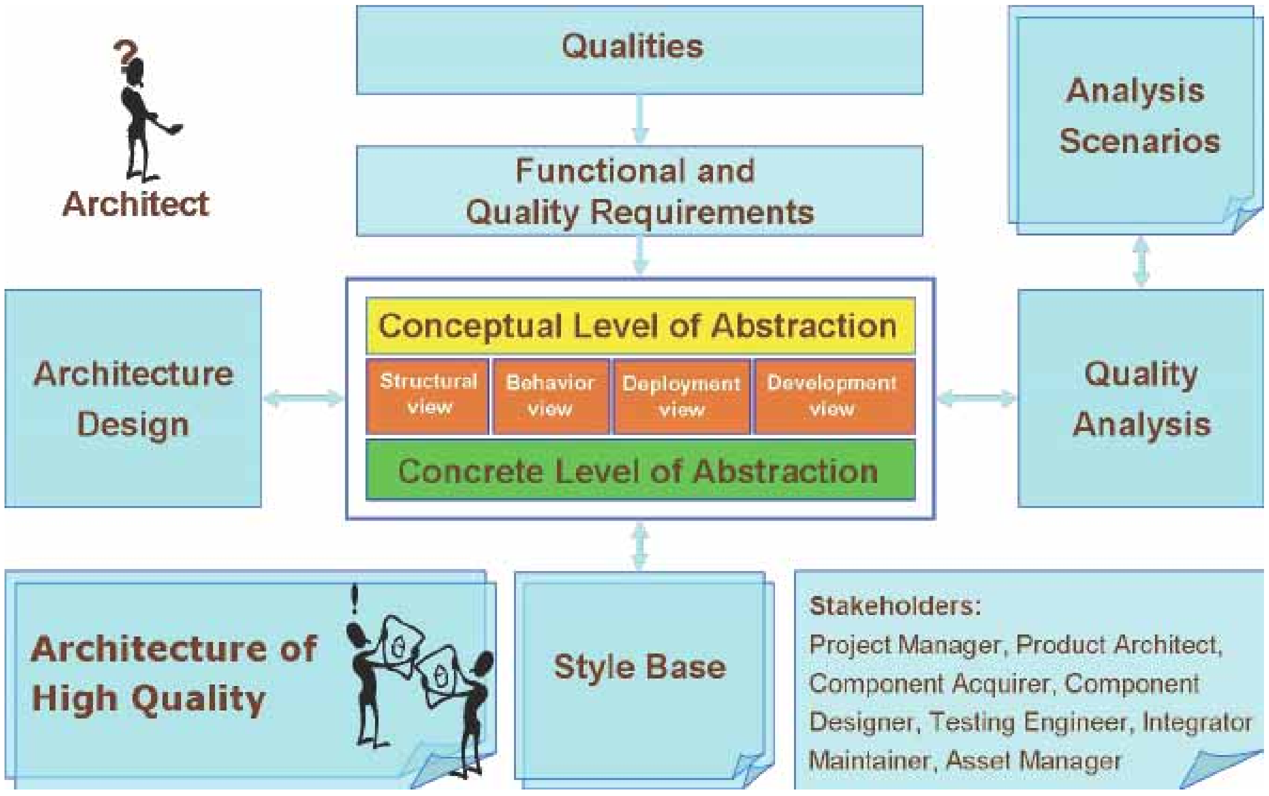
Architect
Qualities
Functional and Quality Requirements
Analysis Scenarios
Architecture Design
Conceptual Level of Abstraction
Structural view
Behavior view
Deployment view
Development view
Concrete Level of Abstraction
Quality Analysis
Architecture of High Quality
Style Base
Stakeholders:
Project Manager, Product Architect, Component Acquirer, Component Designer, Testing Engineer, Integrator Maintainer, Asset Manager

Fig. (1). Abstraction levels and views of QADA® methodology. The conceptual and concrete levels of QADA® consist of four viewpoints.

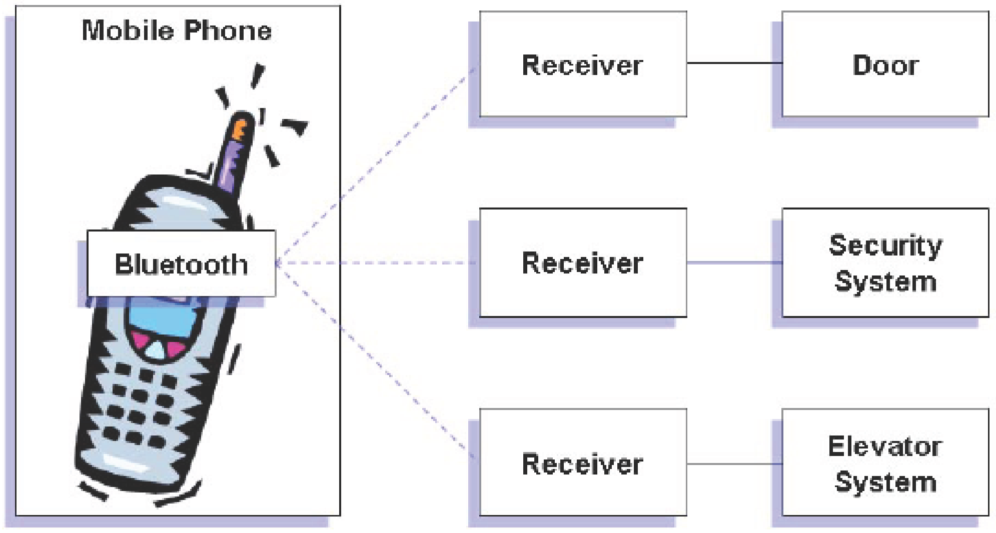

Fig. (2). Overview of the target system.

existing target system, aimed to run on multiple platforms, (2) to advance and improve their current development process for adaptable software system architectures, and (3) to find a more outlined and controlled way of developing software system architectures.

APPLYING THE PHASES TO THE CASE

In this section the phases and steps of AEM, and how they can be applied to the case, are described and discussed.

Phase 1: Defining Adaptability Goals

The purpose of the first phase of AEM is to define adaptability goals. This means identifying and negotiating adaptability requirements to find a satisfactory set that is subsequently brought further into the architecture design. The first phase of AEM includes five steps as follows.

Identifying Stakeholders and Their Concerns

Every system has several stakeholders, i.e. persons involved in system development. Each stakeholder has stakeholders' own interests regarding the system. Stakeholders can also be responsible for a set of activities, such as requirements specification, architecture design, coding or testing. The goals of the system stakeholders' must be in accordance with all of the interest groups of the product. Stakeholders related to the creation and use of architectural descriptions includes the clients, users, architect, developers and evaluators. The requirements of all the stakeholders must be identified and negotiated.

In order to achieve the final adaptability requirements set for the system (i.e. the quality goals), in AEM, adaptability requirements of all the stakeholders can be identified and negotiated by exploiting the i* ("distributed intentionality") framework [43]. The i* framework traces the requirements to stakeholders and their dependency relationships and it helps to detect where the quality requirements originate and what kind of negotiations should take place. In the i* framework stakeholders are represented as (social) actors who depend on each other for goals to be achieved, tasks to be performed, and resources to be furnished. The i* framework includes a graph named Strategic Dependency Model (SDM) for describing the network of relationships and dependencies among the actors. The type of the dependency describes the nature of the agreement as follows: (1) goal dependencies are used to represent delegation of responsibility for fulfilling a goal, (2) softgoal dependencies are similar to goal dependencies, but their fulfillment cannot be defined precisely, (3) task dependencies are used in situations where the actor is required to perform a given activity, and (4) resource dependencies require the actor to provide a resource to the other actor. Normally, in the i* framework actors are represented as circles and dependums, i.e. goals, softgoals, tasks and resources, are respectively represented as ovals, clouds, hexagons and rectangles. Dependencies have the form depender → dependum → dependee.

The NFR (Non-Functional Requirement) framework [5] refines and extends the i* framework. The NFR framework aims to refine the quality requirements, consider different design alternatives, perform trade-off analyses and evaluate the degree to which the requirements are satisfied. The NFR framework utilizes non-functional requirements to drive the overall design process. It assists in acquiring and accessing the required knowledge of the domain and system. The framework identifies the particular NFRs for the domain and the possible design alternatives ("operationalizations") for meeting requirements. It also detects the interdependencies among NFRs and operationalizations, and assists in the selection of the architectural style among operationalization alternatives.

The CBSP (Component-Bus-System-Property) method [44] aims to reconcile the requirements and architectures using intermediate models. The intermediate model is used as a bridge while refining and transforming the requirements to architectural elements. The method defines five steps from the requirement selection to making trade-off choices of architectural elements and styles. Each requirement is assessed for its relevance to the system architecture's components, connectors and topology of the system.

Different sets of quality concerns can be transformed by architecture design into different architectural decisions. Together the NFR framework and CBSP method can be used to define, among other things, how adaptability requirements lead to different architectural decisions. This is valuable for adaptability evaluation. These two approaches are primarily aimed at a one-of-a-kind system development.

In the case implementation, the SDM graph of the i* framework has been exploited to acquire all the change requirements of the stakeholders and their dependency relationships related to the target system. In the SDM graph (Fig. (**3**)), the stakeholders has been represented as (social) actors who depend on each other for goals to be achieved, tasks to be performed, and resources to be furnished. UML2 [45] notation has been exploited to depict the goals, softgoals, tasks and resources of the SDM by using the stereotypes <<actor>> and <<dependency>> and tagged values. The SDM graph has been utilized in order to acquire 19 change requirements relevant to the target system. The change requirements have also been recorded in the table (Table **4**) for future use.

Refining Adaptability Requirements

After the adaptability requirements are identified and negotiated, they must be refined to the final requirements of the system that are considered further in the architecture design. All of the requirements must be provided with the identification numbers. It is not always possible to implement all of the requirements, for example, because time or money. Therefore, the importance of each adaptability requirement must be defined. In AEM, the importance is expressed by using three categories: high, medium and low.

In the case implementation, the specification of the final set of the adaptability change requirements of the target system has been performed by selecting them from the change requirements acquired in the previous step. These requirements have been highlighted in a table (Table **4**) with identification numbers and other importance category information.

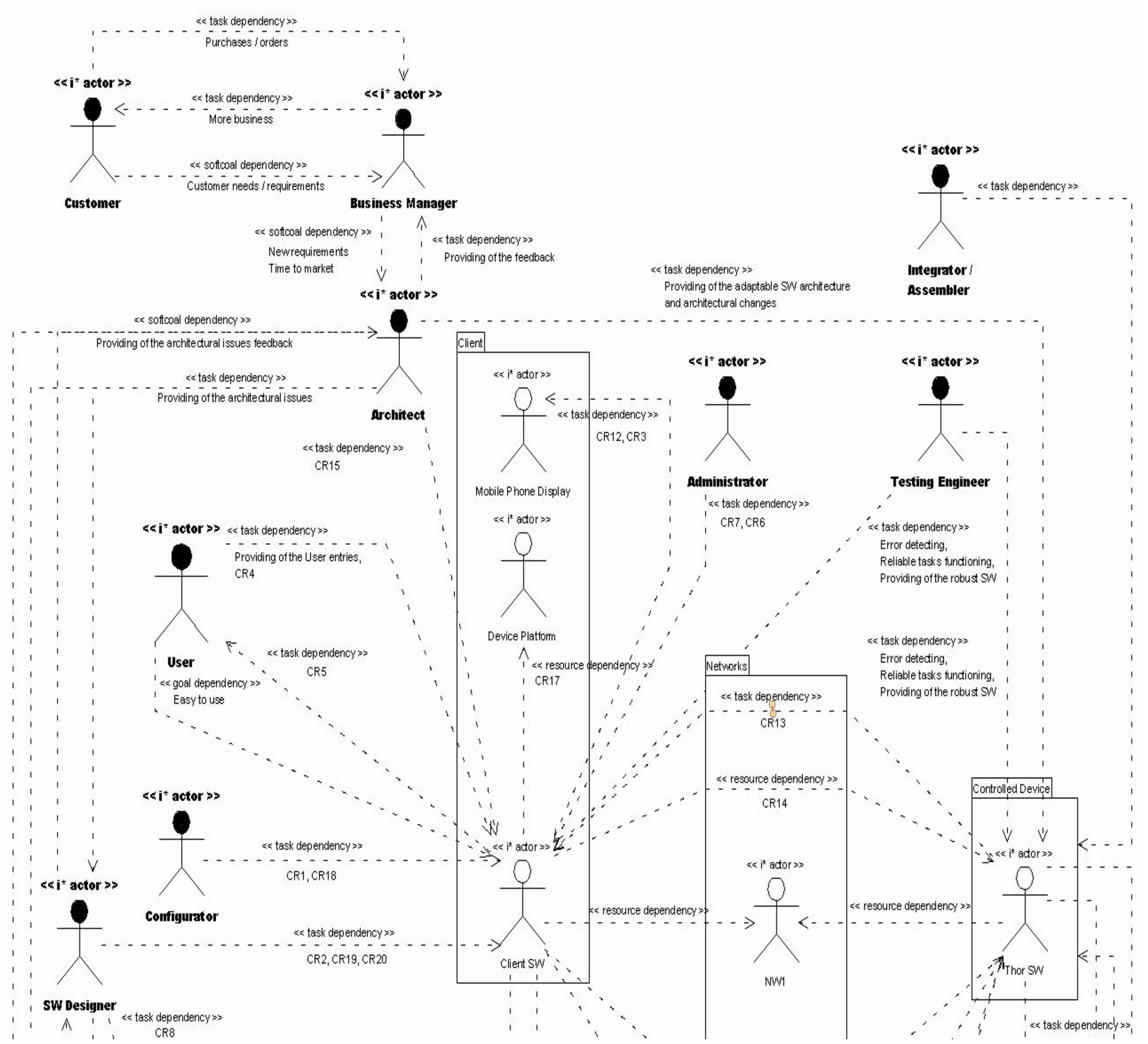

Fig. (3). Partial view of the Strategic Dependency Model (SDM) graph of the target system.

Table 4. Change Requirements for the Target System (Adaptability-Related Requirements are Highlighted)

CR ID	Change requirement description	i* actor (Depender)	i* actor (Dependee)	Type of dependency	Importance
CR1	Add or remove Controlled Device and its UI-icon from the system related to doors, windows, elevators, etc.	Configurator, Integrator	Client SW	Task	High
CR2	Add the new UI-language for the SW developing environment.	SW Designer	Client SW	Task	
CR3	Adapt the UI according to different mobile phone display sizes and models.	Client SW	Mobile Phone Display	Task	High
CR4	Control the UI by the human voice commands.	User	Client SW	Task	
CR5	Inform and guide user by human voice.	Client SW	User	Task	
CR6	Manage user’s privileges and user groups in real-time.	Administrator	Client SW	Task	High
CR7	Change user’s authority.	Administrator	Client SW	Task	High
CR8	Add new software protocol to control the 3rd party devices by means of the serial, USB or CAN interface.	SW Designer	Thor SW	Task	
CR9	Add new hardware interface for serial, USB and CAN interfaces.	HW Designer	Controlled Device	Task	
CR10	Add new baseband hardware for alternative network (NW2).	HW Designer	Controlled Device	Task	
CR11	Provide reliable and secure messaging protocol for Bluetooth and alternative network (NW2) messaging.	Client SW	Thor SW	Task, Recourse	
CR12	Adapt UI in indoor environment based on the location changes of the user’s phone among the surrounding Controlled Devices.	Client SW	Mobile Phone Display	Task	High
CR13	Localize outdoor users by means of Global Positioning System (GPS).	Client SW	GPS	Task, Recourse	
CR14	Send location information of Controlled Device.	Thor SW	Client SW	Task, Recourse	High
CR15	Add architectural changes to the system.	Architect	Client SW, Thor SW	Task	
CR16	Establish external device connection by means of serial, USB or CAN connection.	Thor SW	3rd Party Device	Task, Recourse	
CR17	Adapt to different operating systems and hardware platforms of mobile phones.	Client SW	Device Platform	Recourse	Medium
CR18	Select connection type between alternative network (NW2) and Bluetooth.	Configurator	Client SW	Task	High
CR19	Add new screen size for used mobile phone.	SW Designer	Client SW	Task	High
CR20	Add secure Bluetooth and alternative network (NW2) messaging protocol.	SW Designer	Client SW	Task	

Mapping Adaptability Requirements to Functionality

According to QADA®, the architecture of the system is first described at the conceptual level. The main functionality, i.e. “what the system does”, can be considered as a main force of the conceptual design. In AEM, mapping of the system-specific adaptability requirements to functionality is performed case-specifically. One adaptability requirement may be mapped to several functional blocks. Additionally, adaptability requirements themselves may result in certain functionality. In this phase, the architect only has to decide which services are responsible for the implementation of each of the requirements.

In the case implementation, the target system has first been decomposed into domains, which then have been decomposed into subsystems. Next, the main functionality of the target system has been divided into functional blocks. UML2 notation has been exploited in a graphical presentation. Furthermore, the functional blocks responsible for the implementation of each of the adaptability change requirement have been identified. One or several adaptability change requirements have been mapped to the functional blocks with their identification numbers. The results of the mapping have been depicted in the form of the table (Table **5**).

Selecting Architectural Styles and Patterns and Performing a Trade-Off Analysis

In general speaking, the environment for the systematic synthesis of architecture level adaptation relies on the following concepts: (1) Software Design Methods, (2) Model-Driven Architecture (MDA), (3) Architectural Models, (4) Architectural Styles, (5) Architectural Style Base, (6) Architectural Patterns, (7) Social Patterns, (8) Reflective Architectures, (9) Reflective Middlewares, (10) Runtime Reflection, (11) Adaptive Middleware, (12) Adaptive QoS Management,

Table 5. Mapping the Adaptability Change Requirements to Functionality of the Target System

Domain	Subdomain	Description of main functionality	CR ID
Client	User Interface	• Adds or removes the Controlled Device and its UI-icon for doors, windows, elevators, etc. • Adapts the UI according to location changes of the user's phone among the surrounding Controlled Devices • Adapts the screen size based on used mobile phone • Manages of user's privileges and the user groups in real-time • Changes user's authority information • Activates/deactivates phone's display views • Delivers the system alerts to active view • Receives user entries from phone's keyboard • Keeps up the display data related to the known mobile phones • Keeps up the user profile data • Draws up the "settings view" onto the phone's display • Draws up the "icon view" onto the phone's display	CR1, CR3, CR6, CR7, CR12, CR17, CR18, CR19
	Phone Controller	• Keeps up the phone number database of the target system • Manages the phone number database of the target system • Manages incoming and outgoing phone calls • Keeps up the state of the mobile phone	CR17
	IR Controller	• Handles switching between the IR-configuration files • Parses the needed IR-commands from the given IR-configuration file • Stores the IR-configuration files	CR17
	Thor Controller	• Handles the external adapter commands, • Constructs and holds all the other Bluetooth components of the Controlled Devices • Parses incoming Bluetooth messages • Generates new Bluetooth message • Sends out Bluetooth messages to Bluetooth network • Receives in Bluetooth messages from Bluetooth network • Generates new alternative network (NW2) message • Sends out messages to NW2 • Receives in messages from NW2 network • Keeps up the database related to the known Controlled Devices and their attributes • Manages opening and closing the Bluetooth sockets	CR1, CR12, CR17, CR18
Controlled Device		• Handles Bluetooth and NW2 connections • Sends out location information at regular short time intervals • Delivers actions to the I/O-relays • Stores the system settings while the Controlled Device is turned off	CR12, CR14, CR18

(13) Component Models, (14) Component-Based Architectures, (15) Adaptive Component Interfaces, (16) Connectors, and (17) Unified Modeling Language 2.0 (UML2).

Software systems can be built from one or several architectural styles. For example, even if the main style is layered the blackboard style can still appear in one of the architectural layers. In the beginning of architecture modeling, the dominant architectural style must at least be selected. When the dominant style is decided upon, the other architectural styles and patterns can be selected for the smaller parts of the architecture where they may be beneficial. According to QADA®, the architectural modeling is begun from conceptual architecture. In the conceptual structural view (Fig. (**1**)), the functionality, i.e. services or utilities, are organized according to the selected architectural style.

The different adaptability requirements should be transformed to the design decisions or architectural styles and patterns in a predefined way. The different design alternatives can be searched for, for example, from a style base [46] that represents the mapping between the quality attributes and design decisions. In addition, architectural patterns [47, 48] and social patterns [49] can be transformed to the design decisions. The effect of architectural patterns on quality attributes is discussed in several studies, such as [47, 50-52]. Each architectural pattern helps to achieve specific global system property, for example, adaptability of distributed system.

Social Patterns are idioms inspired by social and intentional characteristics used to design the details of system architecture. Architectures like Multi-Agent Systems (MAS) Architectures appear to be more appropriate than traditional ones for building latest generation software that is typically concurrent, distributed, and dynamic [49]. MAS could be seen as a social organization of autonomous software entities (agents) that can flexibly achieve agreed-upon intentions by interacting with one another. MAS do allow dynamic and evolving structures which can change at runtime to benefit from the capabilities of new system entities or replace obsolete ones. Since the fundamental concepts of Multi-Agent Systems are social and intentional, rather than object, functional or implementation-oriented, the design of MAS Architectures should be eased by using Social Patterns rather than Object-Oriented Design Patterns [49]. A social organization-based MAS development can help matching the system architecture with its operational environment [49]. The choice of these styles and patterns is not necessarily final, but rather iterative, as more exact designs are made later.

Adaptability levels of the systems define how important the achievements of adaptability requirements are to that specific system. For example, in the high adaptability level, adaptability of the system must be guaranteed using the best possible design techniques. The cost and effort of the design is normally higher in the case of high adaptability level systems, whereas in the case of normal and low adaptability level systems the simpler and inexpensive design techniques are used.

There is always a risk that adaptability requirements will conflict with other quality requirements. This might even result in all of the important requirements not being met in the architecture. The purpose of the trade-off analysis is to guarantee the best requirements set considering all of the quality requirements. The NFR framework [5] is one method for the negotiation of various conflicting quality attributes and evaluating the criticality of quality requirements. The NFR framework is a process oriented approach that treats quality requirements as soft goals, i.e. the quality goals, to be achieved [5]. By using the NFR framework, the requirements with the affected stakeholders can be renegotiated and a solution can be found that makes acceptable trade-offs for all of the stakeholders. As a consequence of the trade-off analysis, the resulting problems of the analysis must be identified and solved.

In the case implementation, the architectural styles and patterns suitable for the case purposes have been selected as follows: (1) the Layers Pattern [47] and the Microkernel Pattern [47] to depict the layers of the architectures, (2) General Communication Middleware (GCM) [53] to solve the adaptive middleware issues, (3) a component-based architecture paradigm [54] to depict the architectures, and (4) connectors [55] to depict the relationships between the architecture components. Furthermore, the technological approaches described in Table **6** have been exploited and used in the case implementation.

Because the case has not been defined to consider quality attributes other than adaptability, a detailed trade-off analysis for the target system has not been seen essential and it has not been performed.

Defining Criteria for Adaptability Evaluation

In AEM, adaptability evaluation criteria are categorized into four evaluation levels: Level 1, product line adaptability requirements; Level 2, system-specific adaptability requirements of high importance; Level 3, system-specific adaptability requirements of medium importance; and Level 4, system-specific adaptability requirements of low importance.

In the case implementation, adaptability evaluation criteria for the selected adaptability change requirements of the target system have been defined to be at level 2 or 3.

Phase 2: Representing Adaptability in Architectural Models

The second phase of AEM provides guidelines for how to model adaptability in software architecture in a way that adaptability analysis can be performed directly from the architecture. The abstraction levels of QADA® are used in adaptability modeling in two ways. First, adaptability of the system is described at the conceptual level, and second, adaptability of the system is described at the concrete level. Adaptability appears in architectural models in two ways: (1) adaptability aspects are attached to the architectural elements, and (2) adaptability requirements result in certain design decisions and functionality. The design decisions should be documented; especially the qualitative analysis relies on documented design rationale. The abstraction levels of QADA® are utilized in adaptability modeling. If the design rationale is documented and there is a mapping between each design decision and adaptability requirements, it can be verified that the requirements are met in the architecture level by following the guidelines of third phase of AEM. The second phase of AEM includes the following three steps.

Table 6. Technological Approaches Exploited in the Case Implementation

Technological approach	Description	Usage in the case implementation	General ref.
Software Design Methods	Emphasize • quality requirements, • abstraction levels, • architectural viewpoints, and • architectural views as a driving force when selecting software structures.	QADA® [15]	[23-25]
Architectural Patterns	• Map the quality requirements to architecture design, • Provides a solution for a particular problem and is thus a realization of a style or styles, • Describe how to build a software system and represent the highest-level patterns in the pattern system, • Express fundamental structural organization schemas for software systems	The Layers Pattern [47], The Microkernel Pattern [47]	[47, 48, 52]
Adaptive Middleware	• Is based on underlying components and network services, • Is used to implement adaptive behavior, for example, to deal with performance alternations, security needs, hardware failures, network outages, fault tolerance, etc., • Reflection is used to gather contextual information so that the middleware services can be adapted according to the context of execution	General Communication Middleware, GCM [53]	[31, 56, 57]
Component-Based Architectures	• Provide interaction interfaces between clients, containers, components and concentrators, • Describe the prerequisites for each component, • Specify how some services can be statically plugged into components, • Separate application programming from deployment, • Allow component programmers to give information about which services to use, • Customize the deployment descriptor in order to adapt the component to the specificity of the runtime environment (transaction, persistency, security, database support, etc.)	To depict the architecture	[31, 54, 58]
Connectors	• Are special kind of components that are used to connect components that interact with each other, • Encapsulate component responsibilities and interdependencies for various kinds of collaborations as system level architectural patterns, • Is a lightweight component which functions as a glue of components and induces a low overload, • Support generation of connectors according to the description of elementary services and aspects	To depict the architecture component relations	[31, 55, 59, 60]
Unified Modeling Language (UML2)	• Is a standard and widely accepted modeling language, • The concept of profiles is suitable for the context of adaptability, Adaptability properties could be represented in architecture with the help of a profile tailored especially for this quality attribute, • Can be extended by specific profiles to support certain quality aspects, • Profile is a stereotyped package that contains model elements that have been customized for a specific domain or purpose by extending the meta-model using stereotypes, tagged definitions, and constraints, • Includes the profiles for modeling quality attributes in architecture have been suggested, such as a UML2 profile for Schedulability, Performance and Time, • Includes a profile for modeling the Quality-of-Service and Fault Tolerance	To depict the architecture in the different views, abstraction level and class level	[45]

Mapping Adaptability to Conceptual Architectural Elements

After the architectural style is selected at the conceptual architectural level, adaptability requirements are brought to the architectural models. In this step, the requirements are attached to the architectural elements in the conceptual views of QADA® (Fig. (**1**)). This means that the requirements are transformed to the required responsibilities of the architectural elements, i.e. the components and connectors. In architecture, the required adaptability guides the design of concrete architecture and helps to make the design decisions. By mapping adaptability requirements to the system behavior in behavioral view, the requirements have an influence on the dynamic aspects of the system. The fourth view of QADA®, i.e. the development view, is used in AEM to organize the design work.

The conceptual structural view represents the static relationships of the architectural elements, i.e. components or

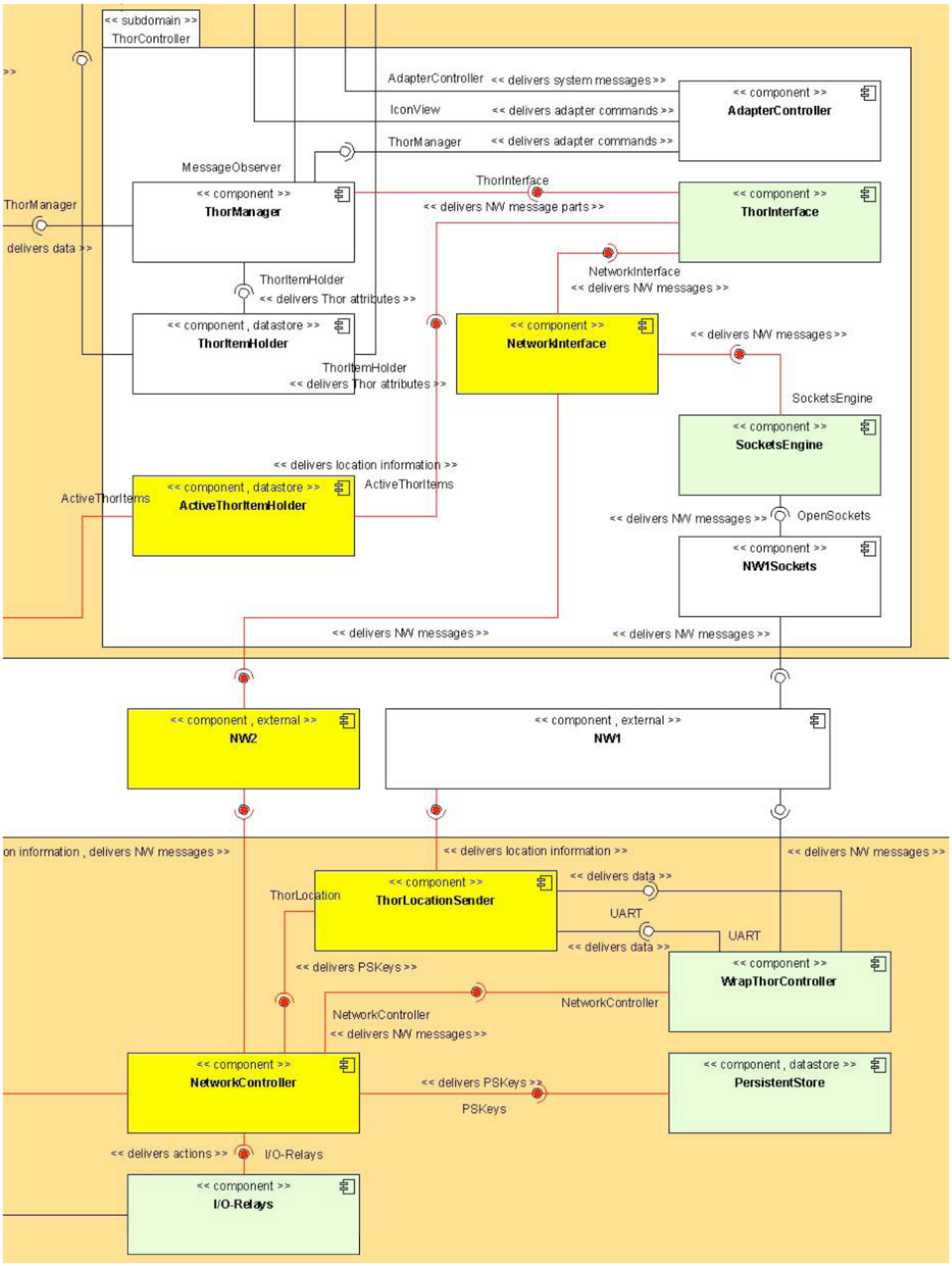

Fig. (4). Partial view of the final component diagram of the target system software architecture with NW2 connection (A3).

services and connectors. The mapping of each adaptability requirement to functionality is performed when defining the quality goals. This enabled the tracing of the requirements to the architecture. Now, vice versa, all adaptability requirements are defined for each architectural element. This enables bidirectional requirements tracing from the architecture to the requirements. By using UML2 notation, the static structure of the system can be represented, for example, by using a component diagram or a composite structure diagram. Typically, the exact means and techniques to implement the requirements are not yet defined, but the definition helps one to define what is required from the system and its elements. Adaptability requirements and design rationale are written inside the architectural elements.

The conceptual behavioral view helps one to understand the dynamic aspects of the system. The view represents the dynamic relationships of the architectural elements. According to QADA®, the behavior of the system is described at the conceptual level as abstract descriptions of collaboration that describe the interactions between the architectural elements. The state diagrams or message sequence diagrams of UML2 notation can be used to describe the interactions between the architectural elements.

The conceptual deployment view allocates units of deployment to physical computing units. In the deployment diagram of UML2, architectural elements are described as deployment nodes or units of deployment with types and relationships. Adaptability is denoted by attaching the requirements to nodes and relationships.

The conceptual development view does not itself assist in adaptability representation. However, the view helps one to detect which architectural elements and services have to be developed, which can be found in the asset repository and the ones that have to be bought.

In the case implementation, the conceptual deployment view for the target system architecture has been designed and depicted by using UML2 notation. The physical computing units have been depicted as nodes in the diagram. The relationships between the nodes have been described and their services have been depicted in a layered style. The Kernel solution has been exploited to hide the used hardware (HW) and the system services from the upper services of the domains. A possible solution for this has been the J2ME [61] or the Microkernel Pattern [47]. GCM has been exploited to provide the adaptive communication middleware solution for the mobile phone Client Software and the Receiver Software of the target system.

Next, the conceptual structural view of the existing target system software architecture (A1) has been represented as a component diagram by using UML2 notation. In a similar manner, the component diagrams for the two new target system software architectures with a Bluetooth connection (A2) and with both an alternative wireless network (NW2) and a Bluetooth connection (A3) (Fig. (**4**)) have been developed and depicted. The adaptability change requirements and design rationale have been written inside the architectural elements, i.e. the components and connectors. The main functionality of the components has been denoted to the diagrams as stereotypes. To depict the required adaptability in diagrams A2 and A3, the necessary new components, the components needed to modify them and the connectors between them have been denoted with different symbol colors. The conceptual behavioral and development views have not been used in the case implementation because they have not seen to bring any additional value to the case work.

Mapping from Conceptual Architecture to Concrete Architecture

When mapping adaptability requirements to the conceptual architecture, the results of adaptability requirements are reflected in the concrete architecture. The traceability of adaptability requirements to the conceptual architecture and the concrete architecture must be ensured. The conceptual architectural elements, i.e. services, are more logical modeling elements than the concrete implementation components. Thus, one conceptual service may result in several concrete components, or one concrete component may contribute to the implementation of one or more conceptual services. The mapping between conceptual and concrete architecture must be documented to trace adaptability requirements to the concrete architectural level. The different design alternatives can be searched again from a style base [46] that represents the mapping between the quality attributes and design decisions. In addition, architectural patterns [47, 48] and social patterns [49] can be transformed to the design decisions.

In the case implementation, the mapping from the conceptual architecture to the concrete architecture has been straightforward: one concrete component responds to one conceptual component. The adaptability-related change requirements have been associated for each architectural component and connector involved in adaptability. In addition, an essential implementation activity, i.e. addition, modification, or deletion, has been associated for each architectural component and connector involved in adaptability. The results of the mapping have been depicted in the form of a table (Table **7**).

Mapping Adaptability to Concrete Architectural Elements

In AEM, adaptability requirements are attached to the architectural components in the concrete views of QADA® (Fig. (**1**)). The concrete view is used in adaptability analysis and is therefore especially tailored to the needs of the analysis. The requirements are attached to architectural elements in the structural and deployment views and are represented in the concrete architecture using the concrete structural and deployment views. In the architecture, adaptability guides the design of concrete components or represents the properties of the existing components, i.e. components in the asset repository or the Commercial Off-The-Shelf (COTS) components, which can be used. The behavioral view assists in the modeling of the behavior of the components and the systems. The development view refines the allocation that is defined in the conceptual development view to concrete components.

The concrete structural view is used to describe the concrete components and interfaces needed for corresponding conceptual architecture. Therefore, the view decomposes the conceptual architecture into lower aggregation levels. The component diagram or composite structure diagram of UML2 notation can be used in order to describe the structure

Table 7. Mapping the Adaptability Change Requirements to the Architectural Elements of the Target System (A3)

Domain	**Sub-domain**	**Component**				**Interfaces**				
		ID and name	**add.**	**mod.**	**del.**	**Name**	**add.**	**mod.**	**del.**	**Change requirement ID**
Client	User Interface	c_1: ViewManager		X		UserEntry				Not relevant in adaptability
						SystemAlerts (PhoneDisplay)				Not relevant in adaptability
						SystemAlerts (MessageObserver)				Not relevant in adaptability
						SettingsView		X		CR1, CR3, CR6, CR7, CR12, CR17, CR18, CR19
						IconView		X		CR1, CR3, CR6, CR7, CR12, CR17, CR18, CR19
						DispaySize	X			CR3, CR17, CR19
						UserProfile	X			CR6, CR7, CR17
						ActiveThorItems	X			CR1, CR12, CR17
		c_4: DispayParameters	X			DispaySize (ViewManager)	X			CR3, CR17, CR19
						DispaySize (SettingsView)	X			CR3, CR17, CR19
						DispaySize (IconView)	X			CR3, CR17, CR19
		c_5: UserProfileHolder	X			UserProfile	X			CR6, CR7, CR17
		c_2: SettingsView		X		SettingsView		X		CR1, CR3, CR12, CR17
						IRController		X		CR17
						DispaySize	X			CR1, CR3, CR12, CR17, CR19
						ActiveView		X		CR1, CR3, CR12, CR17
						ThorItemHolder		X		CR1, CR12, CR17
		c_3: IconView		X		IconView (MessageObserver)		X		CR1, CR3, CR12, CR17
						ContactsArray		X		CR17
						IconView (AdapterController)		X		CR1, CR3, CR12, CR17
						ThorItemHolder		X		CR1, CR12, CR17
						IRController		X		CR17
						DispaySize	X			CR1, CR3, CR12, CR17, CR19
						IconView (ViewManager)		X		CR17
						ActiveView		X		CR1, CR3, CR12, CR17
	Phone Controller	c_6:AddressBook Manager				PhoneBook				CR17
						ConnectCall				CR17
						ContactsArray				CR17

(Table 7). Contd…

		c_7: PhoneCallHandler				ConnectCall				CR17
						PhoneModuleAPI				CR17
						MessageObserver				CR17
	IR Controller	c_8: IRController				IRController (SettingsView)				CR17
						IRController (IconView)				CR17
						ThorManager				CR17
						IRCodePraser				CR17
		c_9: IRCodeParser				IRCodePraser				CR17
						IRConfigFiles				CR17
		c_{10}: IRConfiguration-Files				IRConfigFiles				CR17
	Thor Controller	c_{11}: AdapterController				AdapterController				CR17
						IconView				CR17
						ThorManager				CR17
		c_{12}: ThorManager		X		MessageObserver				CR17
						ThorManager (AdapterController)				CR17
						ThorInterface		X		CR12, CR17, CR18
						ThorItemHolder				CR17
						ThorManager (IRController)				CR17
		c_{13}: ThorInterface		X		ThorInterface		X		CR12, CR17, CR18
						ActiveThorItems		X		CR12, CR17
						NetworkInterface		X		CR17, CR18
		c_{14}: ThorItemHolder				ThorItemHolder (ThorManager)		X		CR17
						ThorItemHolder (SettingsView)		X		CR1, CR12, CR17
						ThorItemHolder (IconView)		X		CR1, CR12, CR17
		c_{15}: NetworkInterface	X			NetworkInterface	X			CR12, CR17, CR18
						SocketsEngine		X		CR12, CR17, CR18
						AirInterface	X			CR12, CR17, CR18
		c_{17}: ActiveThorItem-Holder	X			ActiveThorItems (ThorInterface)	X			CR12, CR17
						ActiveThorItems (ViewManager)	X			CR12, CR17
		c_{16}: SocketsEngine		X		SocketsEngine		X		CR12, CR17
						OpenSockets				CR17
		c_{18}: BluetoothSockets		X		BluetoothSockets		X		CR17
						AirInterface		X		CR17

(Table 7). Contd…

Controlled Device		c_{20}: IR-Sender		X		AirInterface				
						UART		X		Not relevant in adaptability
		c_{22}: NetworkController	X			AirInterface	X			CR12, CR14
						ThorLocation	X			CR12, CR14
						NetworkController	X			CR12, CR14
						PSKeys	X			CR12
						I/O-Relays	X			CR12
						UART	X			Not relevant in adaptability
		c_{19}: ThorLocation Sender	X			ThorLocation	X			CR12, CR14
						AirInterface	X			CR12, CR14
						UART (implement)	X			Not relevant in adaptability
						UART (required)	X			Not relevant in adaptability
		c_{21}: WrapThor Controller		X		AirInterface		X		CR12, CR14
						UART (implement)	X			Not relevant in adaptability
						UART (required)	X			Not relevant in adaptability
						NetworkController		X		CR18
		c_{23}: PersistentStore		X		PSKeys		X		CR12
		c_{24}: 6 x Relay				I/O-Relays				
		c_{25}: I/O-Relays		X		I/O-Relays (NetworkController)		X		Not relevant in adaptability
						I/O-Relays (6 x Relay)				

of the system. The adaptability requirements are attached to the architectural elements by using the same criteria as in conceptual levels. The concrete structural view also reveals the interfaces of the components. Interfaces must be described in a way that enables the estimation of the interoperability of components.

In the concrete behavioral view, the state diagrams or message sequence diagrams of UML2 notation can be used to describe the interactions between components. For each new component, the state diagram must be defined to describe the internal states and state transition. The message sequence diagram is used to derive input messages for adaptability analysis. Also, the activity diagram is required to derive a model for adaptability analysis. An activity diagram typically represents the operational work flows of a system.

The concrete deployment view is used to describe the concrete hardware and software components, the relationships between the hardware components, and the relationships between the software and hardware components. However, AEM concentrates on software systems; therefore this portion is limited.

The concrete development view links the architectural views to the repository of common assets. Thus, the components that already exist can be linked to the concrete components that they realize. When more detailed description of the architecture components and their interfaces are needed, they can be developed and depicted as class diagrams.

In the case implementation, the mapping from the conceptual architecture to the concrete architecture has been straightforward; one concrete architectural element corresponds to one conceptual architectural element. However, more detailed description of the architecture components and their interfaces has been needed. The components and their interfaces, related to the adaptability change requirements, have been developed and depicted as class diagrams by using UML2 notation (an example is given in Fig. (**5**)).

Furthermore, adaptability of the components has been realized by the structure of their interfaces and the communication messaging abstraction between them. For example, GCM (Fig. (**6**)) consists of two main parts: message structure and middleware for processing the messages providing a communication abstraction for application messaging [53]. The main rationale for introducing the communication abstraction for application messaging is to enhance adaptability and facilitate development of the application. GCM is designed for processing GCM messages. It facilitates the development of the distributed applications by providing to

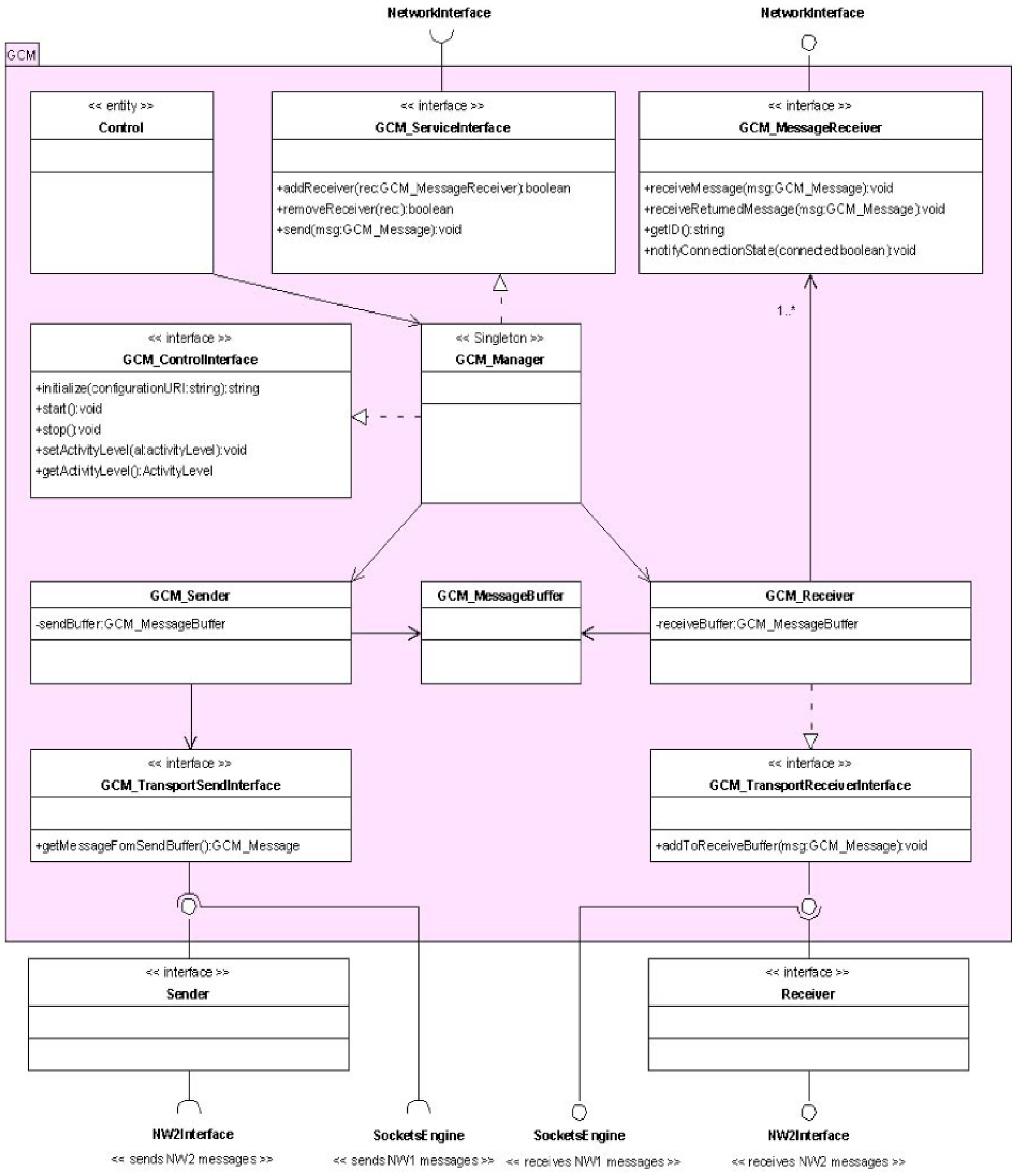
NetworkInterface
NetworkInterface
GCM
<< entity >>
Control
<< interface >>
GCM_ServiceInterface
+addReceiver(rec:GCM_MessageReceiver):boolean
+removeReceiver(rec:):boolean
+send(msg:GCM_Message):void
<< interface >>
GCM_MessageReceiver
+receiveMessage(msg:GCM_Message):void
+receiveReturnedMessage(msg:GCM_Message):void
+getID():string
+notifyConnectionState(connected:boolean):void
1..*
<< interface >>
GCM_ControlInterface
+initialize(configurationURI:string):string
+start():void
+stop():void
+setActivityLevel(al:activityLevel):void
+getActivityLevel():ActivityLevel
<< Singleton >>
GCM_Manager
GCM_Sender
-sendBuffer:GCM_MessageBuffer
GCM_MessageBuffer
GCM_Receiver
-receiveBuffer:GCM_MessageBuffer
<< interface >>
GCM_TransportSendInterface
+getMessageFomSendBuffer():GCM_Message
<< interface >>
GCM_TransportReceiverInterface
+addToReceiveBuffer(msg:GCM_Message):void
<< interface >>
Sender
<< interface >>
Receiver
NW2Interface
<< sends NW2 messages >>
SocketsEngine
<< sends NW1 messages >>
SocketsEngine
<< receives NW1 messages >>
NW2Interface
<< receives NW2 messages >>

Fig. (5). Class diagram of the Network Interface (c_{15}).

adopt middleware communication service removing the need for application specific communication implementations [53]. The difference between GCM and other known communication middleware solutions is that GCM constitutes a transport-independent communication abstraction, making it widely reusable in a variety of applications [53]. GCM solution of the target system software architecture has been message oriented and GCM messages have been defined in object oriented fashion. GCM message structure has been dynamic consisting of message body (GCM_Message) and message elements (GCM_MessageElement) [53]. New message elements for different kind of data have been introduced to GCM message structure by extending GCM_MessageElement -class [53]. GCM messages use identifiers of the target system software for addressing. These are simply text format IDs that will be compared for correct delivery at the middleware [53].

GCM of the target system software architecture has been defined in UML2 as reusable entity for processing GCM messages. GCM provides one interface to utilize, and one to implement. GCM implements the receiver interface (GCM_Message Receiver) for receiving messages, connection state information, and returned messages in case of unsuccessful sending [53].

The receiver interface has been utilized as call-back interface by GCM and must therefore be registered at the middleware. The application is able to register or unregister object instances implementing the receiver interface by using the service interface (GCM_ServiceInterface) [53]. The service interface also contains the main functionality of GCM, i.e. message sending.

GCM of the target system architecture shares similarities with most of the communication software including the same basic functionality like send and receive buffers. In case of sudden disconnection the send buffer can be cleared and all unsent GCM messages can returned to the application *via* the receive interface [53]. If required, parallelism within the middleware may be provided by using worker threads and dynamic thread pool within GCM_Receiver -component for delivering received GCM messages [53]. This way the deadlocks caused by blocking applications and single thread delivery can be avoided [53].

GCM architecture contains two interfaces for achieving the data transport independence; one for sending messages into the network (GCM_ TransportSendInterface) and another for receiving them from the network (GCM_TransportReceive Interface) [53]. With these interfaces GCM is able to accommodate transport implementations using e.g. TCP/IP (Transmission Control Protocol/Internet Protocol) for transfer of the messages. The different communication paradigms are not reflected to GCM middleware architecture making GCM middleware communication paradigm transparent and leaving the selection of applied communication paradigm and connection management for the transport implementation [53].

The concrete behavioral and development views have not been used in the case implementation because they have not been seen to bring any additional value for the case work.

Phase 3: Evaluation of Adaptability

The third phase of AEM is about analyzing the candidate architectures to validate whether or not adaptability require-

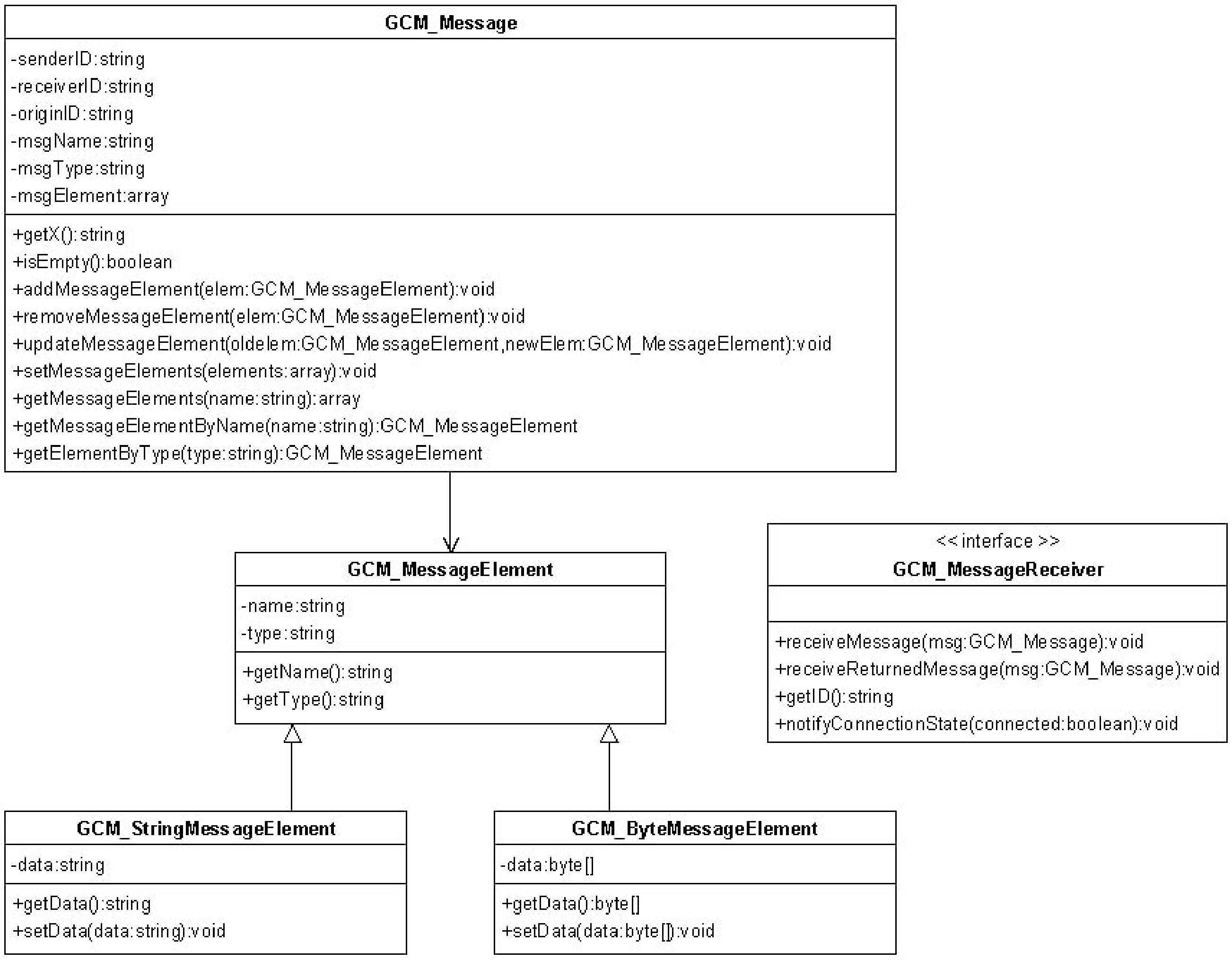
GCM_Message
-senderID:string
-receiverID:string
-originID:string
-msgName:string
-msgType:string
-msgElement:array
+getX():string
+isEmpty():boolean
+addMessageElement(elem:GCM_MessageElement):void
+removeMessageElement(elem:GCM_MessageElement):void
+updateMessageElement(oldelem:GCM_MessageElement,newElem:GCM_MessageElement):void
+setMessageElements(elements:array):void
+getMessageElements(name:string):array
+getMessageElementByName(name:string):GCM_MessageElement
+getElementByType(type:string):GCM_MessageElement
GCM_MessageElement
-name:string
-type:string
+getName():string
+getType():string
<<interface>>
GCM_MessageReceiver
+receiveMessage(msg:GCM_Message):void
+receiveReturnedMessage(msg:GCM_Message):void
+getID():string
+notifyConnectionState(connected:boolean):void
GCM_StringMessageElement
-data:string
+getData():string
+setData(data:string):void
GCM_ByteMessageElement
-data:byte[]
+getData():byte[]
+setData(data:byte[]):void

Fig. (6). Class Diagram of GCM message structure.

ments are met. Adaptability evaluation is performed by using quantitative and qualitative analyses.

The quantitative adaptability analysis evaluates the architecture adaptability based on adaptability scenario profile and impact analysis of a system based on its structure in terms of composition, i.e. components and their interactions. This analysis requires that the structure of the system is known, both the static aspects represented by its components and the dynamic aspects represented by the execution frequency of each component and each interaction between components. The quantitative approach also assumes that the behavior of the components and component interactions are known.

Qualitative analysis is complementary to the quantitative one and can be applied without knowing the behavior of components. The analysis consists of reasoning the design decisions, e.g., architectural styles and patterns and their support for adaptability requirements. Finally, decision making is performed based on the analyses.

Quantitative Analysis

The purpose of the quantitative analysis is to provide a systematic adaptability evaluation approach to support architecture improving and decision making for choosing among candidate architectures.

AEM adopts the Quantitative Evaluation Approach introduced in [28] for evaluation of quantitative adaptability of the architectures. In AEM, quantitative analysis is driven by stakeholders' adaptability goals, including the following four steps: (1) developing an Adaptability Scenario Profile (ASP) for each of the candidate architecture, based on the system's adaptability goals, (2) performing an impact analysis under the scenario profile, (3) applying the metric and calculating the value of adaptability degree, and (4) analyzing the results of adaptability evaluation.

Scenarios are one of the effective techniques in architecture analysis [28, 62]. An adaptability scenario covers a typical use for the system related to adaptability, system behavior for abnormal conditions and potential future changes to the system. Adaptability scenario is a description of the system behavior driven by the change requirement, including the use of the system, the reaction to the change requirement and potential future changes, all of which are related to adaptability. The ASP is a set of related adaptability scenarios.

In the case implementation, the ASP of the target system has been developed by defining the scenarios for all the adaptability-related change requirements and recording them in the table (Table **8**) with the change requirement identification, the related information on the scenario identification (S_k), the estimated probability of the scenario (PS_k), scenario description, and the evaluation criteria.

Once the ASP has been available, the impact analysis for each of the adaptability scenarios has been performed by exploiting the Class Point [63] (CP) method and the Class Point Calculation Worksheet (CPCW) (Table **13**).

In the CP size estimation, the design specifications have first been analyzed in order to identify and classify the classes [63]. The Problem Domain Type (PDT) component contains classes representing real-world entities in the system's application domain. The classes of the Human Interaction Type (HIT) are designed to satisfy the need for informa-

Table 8. Adaptability Scenario Profile (ASP) for the Target System

Change req. ID	S_k	PS_k	Scenario description	Evaluation criteria
CR1	S_1	0,15	Configurator of the system adds or removes the information related to the Controlled Device (i.e. doors, windows, elevators, etc.) to or from the configuration file of the system by sending the change request message to the Client Software. At the same time, Configurator adds or removes the Controlled Device UI-icon on or from the Mobile Phone Display by sending the change request message to the Client Software.	Level 2
CR3, CR19	S_3	0,15	The Client SW adapts the mobile phone display size by sending the change request message to the Mobile Phone Display and by using the display size parameters of the mobile phones added by SW Designer.	Level 2
CR6	S_6	0,15	Administrator of the system manages the user's privileges and the user groups of the system in real-time by sending the related information to the user profile files stored by the Client SW.	Level 2
CR7	S_7	0,15	Administrator of the system maintains the user's authority information in real-time by sending the related information to the user profile files stored by the Client SW.	Level 2
CR12, CR14	S_{12}	0,15	In indoor environment, the Client SW adapts in real-time the information related to the Controlled Devices on the user's mobile phone display by receiving the location information messages from the Controlled Devices and by sending the change request messages to the Mobile Phone Display. The location information message is sent in real-time by the Controlled Device at regular short intervals and when the range of the Bluetooth or the alternative network connection is valid.	Level 2
CR17	S_{17}	0,10	The Client SW adapts to the different OS and HW platforms of the mobile devices by using layered architecture and the Kernel solution to hide the Client SW from the OS and the HW.	Level 3
CR18	S_{18}	0,15	Configurator of the system selects the connection type between alternative network and Bluetooth by sending the corresponding request message to the Client SW.	Level 2

Table 9. Measuring Criteria for the Target System Architecture (A3)

Domain	Sub-domain	Component						Interfaces		
		ID and name	Number of classes				NSR	Name	NEM	NOA
			PDT	HIT	DMT	TMT				
Client	User Interface	c_1:ViewManager	0	0	0	2	8	UserEntry	1	0
								SystemAlerts (PhoneDisplay)	1	0
								SystemAlerts (MessageObserver)	1	0
								SettingsView	3	1
								IconView	3	1
								DispaySize	7	0
								UserProfile		
								ActiveThorItems		
		c_4:DispayParameters	0	0	1	1	3	DispaySize (ViewManager)	7	0
								DispaySize (SettingsView)		
								DispaySize (IconView)		
		c_5:UserProfileHolder	0	0	1	1	1	UserProfileHolder	7	0
		c_2:SettingsView	0	1	1	4	5	SettingsView	4	0
								IRController	1	0
								DispaySize	7	0
								ActiveView	2	0
								ThorItemHolder	3	0
		c_3:IconView	0	1	1	5	8	IconView (MessageObserver)	1	0
								ContactsArray	2	0
								IconView (AdapterController)	7	0
								ThorItemHolder	4	0
								IRController	2	0
								DispaySize	7	0
								IconView (ViewManager)	4	0
								ActiveView	2	0
	Phone Controller	c_6: AddressBook Manager	0	0	1	1	3	PhoneBook	4	1
								ConnectCall	3	0
								ContactsArray	3	0
		c_7:PhoneCallHandler	0	0	0	1	3	ConnectCall	3	0
								PhoneModuleAPI	4	1

(Table 9). Contd…..

								MessageObserver	1	0
	IR Controller	c_8:IRController	0	0	2	4	4	IRController (SettingsView)	2	0
								IRController (IconView)	2	0
								ThorManager	3	0
								IRCodePraser	3	0
		c_9:IRCodeParser	0	0	0	2	2	IRCodePraser	2	0
								IRConfigFiles	2	0
		c_{10}:IRConfiguratio Files	0	0	1	1	1	IRConfigFiles	3	0
	Thor Controller	c_{11}:Adapter Controller	0	0	1	1	3	AdapterController	4	0
								IconView	3	0
								ThorManager	3	0
		c_{12}:ThorManager	0	0	1	2	5	MessageObserver	1	0
								ThorManager (AdapterController)	3	0
								ThorInterface	7	0
								ThorItemHolder	5	0
								ThorManager (IRControl-ler)	3	0
		c_{13}:ThorInterface	0	0	0	2	3	ThorInterface	3	0
								ActiveThorItems	7	0
								NetworkInterface	7	0
		c_{14}:ThorItemHolder	1	0	1	2	3	ThorItemHolder (ThorManager)	3	0
								ThorItemHolder (SettingsView)	10	0
								ThorItemHolder (Icon-View)	10	0
		c_{15}:NetworkInterface	0	0	3	13	3	NetworkInterface	7	0
								SocketsEngine	2	0
								AirInterface	6	0
								GCM_Message	15	10
		c_{17}:ActiveThorItem Holder	1	0	1	2	3	ActiveThorItems (Thor-Interface)	7	0
								ActiveThorItems (ViewManager)	10	0
		c_{16}:SocketsEngine	0	0	0	1	2	SocketsEngine	7	0
								OpenSockets	7	0
		c_{18}:BluetoothSockets	0	0	0	2	2	BluetoothSockets	7	0
								AirInterface	3	0

(Table 9). Contd…..

Controlled Device		c_{20}:IR-Sender	0	0	0	1	2	AirInterface	1	0
								UART	4	0
		c_{22}:Network Controller	0	0	3	13	6	AirInterface	6	0
								ThorLocation	2	0
								NetworkController		0
								PSKeys	7	0
								I/O-Relays		0
								UART		0
								GCM_Message	15	10
		c_{19}:ThorLocation Sender	0	0	1	2	4	ThorLocation	7	0
								AirInterface	6	0
								UART (implement)	2	0
								UART (required)	2	0
		c_{21}:WrapThor Controller	0	0	1	2	3	AirInterface	7	0
								UART (implement)	7	0
								UART (required)		0
								NetworkController		0
		c_{23}:PersistentStore	0	0	1	1	1	PSKeys	7	0
		c_{24}:6 x Relay	0	0	1	1		I/O-Relays	1	0
		c_{25}:I/O-Relays	0	0	1	1	2	I/O-Relays (NetworkController)	7	0
								I/O-Relays (6 x Relay)	6	0

tion visualization and human interaction. The Data Management Type (DMT) component encompasses the classes that offer functionality for data storage and retrieval. The Task Management Type (TMT) classes are designed for purposes of task management and communication between subsystems and external systems.

Second, the behavior of each class has been taken into account in order to evaluate its complexity level (low, average or high). The Number of External Methods (NEM) measures the size of the interface of a class and is determined by the number of locally defined public methods [63]. The Number of Services Requested (NSR) provides a measure of the interconnection of the system components and is determined by the number of different services requested from other classes [63]. The Number of Attributes (NOA) has also been taken into account in order to evaluate the complexity level of a class [63]. For example, if a class had more than nine NEM and the NSR value was not less than 2, and NOA was bigger than or equal to ten, a high complexity level has been assigned.

In the case implementation, the CP measuring criteria for each component and the connector of the candidate architectures A1, A2 and A3, has been collected and recorded in form of a table (an example is given in Table **9**).

Once the complexity level of each identified class has been established, the Total Unadjusted Class Point (*TUPC*) has been computed as the weighted total of the four components of the application [63]:

$$TUCP = \sum_{i=1}^{4} \sum_{j=1}^{3} w_{ij} \times x_{ij}$$

where x_{ij} is the number of classes of component type *i* (PDT, HIT, DMT, TMT) with complexity level *j*, and w_{ij} is the empirical weighting value for type *i* and complexity level *j*.

Next, the Technical Complexity Factor (*TCF*) has been determined by assigning the degree of influence (ranging from 0 to 5) that 18 general system characteristics had on the application [63]. The estimates given for the degrees of influence have been recorded in the Processing Complexity Table in the CPCW (Table **13**). The sum of the influence degrees related to such general system characteristics forms the Total Degree of Influence (*TDI*), which is used to determine the *TCF* based on the following formula [63]:

$$TCF = 0{,}55 + (0{,}01 \times TDI)$$

The final value of the *CP* for each component and connectors has been obtained by multiplying the *TUPC* value by *TCF* [63]:

$$CP = TUCP \times TCF$$

Two metrics have been used for architecture adaptability: *IOSA* (Impact On the Software Architecture) and *ADSA* (Adaptability Degree of the Software Architecture) to measure the impact on the architectures [28]. The *IOSA* has been calculated by summing the *CP* values in each adaptability scenario [28]:

$$IOSA = \sum_{k=1}^{|S|} PS_k \times (CP(C_{sk}) + CP(T_{sk})) \quad (1)$$

where |S| is the number of adaptability scenario, PS_k is the estimated probability of adaptability scenario S_k based on adaptability evaluation criteria, *CP* is the impact analysis result of S_k, and C_{sk} and T_{sk} are the set of impacted components and connectors S_k respectively.

The application of the *IOSA* is that the degree of adaptability has an inverse relationship to the value of *IOSA*, so the *ADSA* is defined as [28]:

$$ADSA = N^{-IOSA}, N > 1 \quad (2)$$

From equation 1, the range of *IOSA* is [0,∞], so the range of *ADSA* is [1,0], 1 means that the architecture is totally adaptable in all dimensions and 0 means that architecture is not adaptable to any change requirement. In order to make the value of *ADSA* spread equally throughout the range, the value of *N* must be close to 1. After some experiments, we define *N* = 1.01. Based on the value of *ADSA* the architect can decide which candidate architecture is more adaptable to stakeholders' adaptability goals and in which dimensions the architecture is adaptable. In addition, the architect can identify the weaknesses of the architectures to support architecture improvement. However, the architectures must be designed for same system or the value of *ADSA* is meaningless.

In the case implementation, the impact analysis has been decided to perform in two phases; firstly without the NW2 connection and secondly with it. Table **10** summarizes the results of the impact analysis for the architecture A1 based on the required changes of the architecture A2. The scenario S_{18} and the connector $t_{15\text{-}22}$ have not been realized in this phase. Table **11** summarizes the results of the impact analysis for the architecture A2 based on the required changes of

Table 10. Impact Analysis and ADSA for the Target System Software Architecture (A1)

Scenario	***PS***	***CP***	**Target of the component changes**			**The target of the connector changes**			***IOSA***	***N***	***ADSA***
			c(add)	**c(mod)**	**c(del)**	**t(add)**	**t(mod)**	**t(del)**			
S_1	0,15	105,66		c_1 c_2 c_3			$t_{1\text{-}2}$ $t_{1\text{-}3}$		15,85		
S_3	0,15	115,29	c_4	c_1 c_2 c_3		$t_{1\text{-}4}$ $t_{2\text{-}4}$ $t_{3\text{-}4}$	$t_{1\text{-}3}$		17,29		
S_6	0,15	72,72	c_5	c_1 c_3		$t_{1\text{-}5}$	$t_{1\text{-}3}$		10,91		
S_7	0,15	72,72	c_5	c_1 c_3		$t_{1\text{-}5}$	$t_{1\text{-}3}$		10,91		
S_{12}	0,15	524,84	c_{15} c_{17} c_{19} c_{22}	c_1 c_3 c_{12} c_{13} c_{16} c_{20} c_{21} c_{23} c_{25}		$t_{13\text{-}17}$ $t_{19\text{-}22}$ $t_{15\text{-}22}$*)	$t_{1\text{-}3}$ $t_{1\text{-}17}$ $t_{13\text{-}12}$ $t_{12\text{-}15}$ $t_{15\text{-}16}$ $t_{22\text{-}20}$ $t_{22\text{-}23}$ $t_{22\text{-}25}$		78,73		
S_{17}	0,10	0,00	The Kernel solution realizes the scenario S_{17}						0,00		
S_{18}	0,15	0,00	The scenario S_{18} is realized in the next phase						0,00		
	1,00	891,23							133,68	1,010	0,264
*) *NW2 connector* $t_{15\text{-}22}$ *is realized in the next phase.*											

Table 11. Impact Analysis and ADSA for the Target System Software Architecture (A2)

Scenario	*PS*	*CP*	Target of the component changes			The target of the connector changes			*IOSA*	*N*	*ADSA*
			c(add)	c(mod)	c(del)	t(add)	t(mod)	t(del)			
S_1	0,15	0,00							0,00		
S_3	0,15	0,00							0,00		
S_6	0,15	0,00							0,00		
S_7	0,15	0,00							0,00		
S_{12}	0,15	19,04		c_{15} c_{22}		$t_{15\text{-}22}$			2,86		
S_{17}	0,10	0,00							0,00		
S_{18}	0,15	56,16		c_1 c_2			$t_{1\text{-}2}$		8,42		
	1,00	75,20							11,28	1,010	0,894

the final architecture A3. The scenario S_{18} and the connector $t_{15\text{-}22}$ have now been realized.

In the case implementation, both of the metrics *IOSA* and *ADSA* have been calculated to measure the impact on the target system architectures A1, A2, and A3. The results of the quantitative analysis are shown in Table **12**.

The results reveal that the architecture A1 is poorly adaptive in the required dimensions described in Table **4**. The architecture A2 is adaptive in required dimensions of CR1, CR3, CR6, CR7, CR12, CR14, CR17, and CR19, but not in CR18 because the connector $t_{15\text{-}22}$ has not been implemented yet. However, the architecture A2 is feasible when NW2 connection has not been required to implement. If NW2 connection is decided to implement later, it will be a relatively easy task because the value of *ADSA* is on high level. When the case is an issue, it's obvious, that the value for *ADSA* for the final architecture A3 is one, meaning that the architecture is adaptive in all the required dimensions.

Qualitative Analysis

The qualitative analysis is about tracking adaptability requirements. In AEM, the qualitative analysis relies on documented design rationale that must be included or accompanied in the architectural models. If this is not the case, then the analysis relies heavily on the architects' tacit knowledge. The process of qualitative analysis can be partly automated, for example by automating the report generation. The main parts of the analysis still require a human analyzer. The bidirectional requirement tracking is about tracking the requirements to the architecture and the properties of the architecture to the requirements. The tracking is performed based on the requirement numbers that are associated to architectural elements by using adaptability profiles. Adaptability profile maps the requirements to the architecture at the conceptual level and describes how these requirements are taken into account at the concrete level. Therefore, the qualitative analysis verifies that each requirement has been taken into account in the architecture design. When analyzing the architecture and its components, the tracking is performed vice versa; from concrete architecture to the conceptual and furthermore to requirements.

Design rationale can be associated with individual components, with individual connections, and a set of components and their connections. The analyzer compares the design decisions with adaptability requirements and analyzes how those requirements are met in the architecture. The analyzer also has to decide if the requirements are met sufficient enough, and to examine how to meet requirements better and how well all of these decisions work together. For comparing two different architectures, the qualitative analysis must be performed for each of the designs, and thereafter a numerical indicator for the coverage of requirements is used, but also human judgment regarding the proposed solutions has to be applied.

The objective of the qualitative analysis is to determine and to identify problems that may occur when certain adapt-

Table 12. Results of the Quantitative Analysis of the Target System Software Architectures

Architecture	Description	Dimensions	ADSA
A1	The Target System (existing architecture)	Non	0,264
A2	The Target System (without NW2 connection)	CR1, CR3, CR6, CR7, CR12 ($t_{15\text{-}22}$ is not implemented), CR14, CR17, CR19	0,894
A3	The Target System (with NW2 connection)	CR1, CR3, CR6, CR7, CR12, CR14 CR17, CR18, CR19	1,000

Table 13. Class Point Calculation Worksheet

Class Point Calculation Worksheet								**Date:**		**dd.mm.yyyy**
Application:	**The Target System**									
Company:	**A Ltd.**									
Component:	**NetworkInterface / A1 / without NW2 connection**									
Component ID:	***C_{15}***									
Prepared by:	**PVT**									
Reviewed by:	**JMA**							**Date:**		**dd.mm.yyyy**
Class Point Count										
Class Type of System Component		**Complexity Level**								
		Low			**Average**			**High**		**Total**
PDT	Problem Domain	**0**	3	0	**0**	6	0	**0**	10	0	0
HIT	Human Interaction	**0**	4	0	**0**	7	0	**0**	12	0	0
DMT	Data Management	**0**	5	0	**0**	8	0	**3**	13	39	39
TMT	Task Management	**0**	4	0	**0**	6	0	**11**	9	99	99
	TUCP	**Total Unadjusted Class Point**								**138**

Processing Complexity (PC)					
ID	**General System Characteristic**	**DI**	**ID**	**General System Characteristic**	**DI**
PC1	Data Communications	5	PC10	Reusability	5
PC2	Distributed Functions	0	PC11	Installation Ease	5
PC3	Performance	5	PC12	Operational Ease	5
PC4	Heavily Used Configurations	0	PC13	Multiple Sites	5
PC5	Transaction Rate	5	PC14	Facilitation of Change	4
PC6	Online Data Entry	0	PC15	System Survivability	5
PC7	End-User Efficiency	0	PC16	Rapid Prototyping	5
PC8	Online Update	5	PC17	Multi-user Interactivity	5
PC9	Complex Processing	5	PC18	Multiple Interfaces	0
			TDI	Total Degree of Influence	64
Technical Complexity Factor (*TCF*) and Class Point Measure (*CP*)					
TCF	**Technical Complexity Factor (= 0.55 + (0.01 x TDI)**				**1,19**
CP	**Class Point Measure (= TUCP x TCF)**				**164,22**

Evaluation criteria for Complexity Level and Processing Complexity

NSR:	2			
NEM:	24			
NOA:	10			
0-2 NSR	0-5 NOA	6-9 NOA	≥10 NOA	NSR = Number of Services Requested
0-4 NEM	Low	Low	Average	NEM = Number of External Methods
5-8 NEM	Low	Average	High	NOA = Number of Attributes
≥ 9 NEM	Average	High	High	
3-4 NSR	0-4 NOA	5-8 NOA	≥ 9 NOA	
0-3 NEM	Low	Low	Average	Degree of Influence (DI) values:
4-7 NEM	Low	Average	High	0 = Not present or no influence
≥ 8 NEM	Average	High	High	1 = Insignificant influence
≥ 5 NSR	0-3 NOA	4-7 NOA	≥ 8 NOA	2 = Moderate influence
0-2 NEM	Low	Low	Average	3 = Average influence
3-6 NEM	Low	Average	High	4 = Significant influence
≥ 7 NEM	Average	High	High	5 = Strong influence, throughout

ability requirements are not met in the architecture. Thus, architect must pay attention to the parts of the architecture that require an enhancement to meet adaptability requirements in this particular architecture, without changing the architectural style.

In the case implementation, the architectures A2 and A3 have been designed in a component, layered, and Object Oriented (OO) fashion, and they both exploit GCM architecture to provide an adaptive middleware solution. The Kernel solution for both of the architectures has also been applied to provide platform independence of the Client Software of the target system.

Decision Making Based on the Analysis

If the result of the qualitative and/or quantitative adaptability analysis reveals that the particular architecture is not sufficient enough for adaptability requirements, the architect has two choices: (1) keep the architecture and increase adaptability of components and their interactions. This can be performed by choosing components with higher adaptability, by implementing higher adaptable components by eliminating software defects in their implementation, and by deploying software on more adaptable hardware, and (2) change the architecture by using different architectural styles and patterns, and by introducing new adaptability mechanisms.

AEM enables adaptability analysis to be performed systematic and iterative way for each architectural choice. The results of the analysis of different architectural choices must be evaluated against adaptability evaluation criteria and against each other. Human analysis is required to decide which architectural alternative meets the requirements best.

In the case implementation, the architecture A2 is feasible when the alternative wireless network connection is not required to be implemented because the value of *ADSA* is on a high level. The final architecture, A3, is useful when both of the network connections are required to be implemented.

CONCLUDING REMARKS

Adaptability, related to software engineering, is considered with different terms (Table **1**). The runtime adaptability is the ability of a software system to adapt itself to changes that occur either internally or externally in its operating environment. A software system can either change its behavior or its structure. The dynamic adaptability of a software system is the system ability to adapt the behavior of applications to the alternations in their environment without reconfiguration. The dynamic adaptability is especially suitable when fast and frequent reactions are required. Dynamic adaptation of runtime software system depends on monitoring, interpretation, and reconfiguration. A reflective system maintains at the runtime data structures that materialize some aspects of the system itself. Reflection means that an explicit, runtime representation of system behavior is maintained, which reflects the actual system behavior in the sense that changes in the latter are materialized in the meta-level description. Self-management is the ability of a software system to be efficient without user intervention.

Quality of software is one of the major issues in software intensive systems and it is important to analyze it as early as possible. It is widely accepted that quality requirements for the final software system can be determined at the software architecture level by means of the quality attributes. The operational quality attributes (Table **2**) are characteristics of the system in operation, e.g. performance, reliability, and robustness. The development quality attributes (Table **3**) are characteristics of the system that are relevant from a software engineering perspective, e.g. maintainability, reusability, and flexibility. Adaptability concerns the whole life cycle of software system, and therefore, it exists at all abstraction levels in software development. In these dimensions adaptability means different things, and therefore, techniques to achieve it also vary.

Adaptability has many facets and it is defined with different ways. In the discussed definition for adaptability of software system and software architecture, adaptability is related to the objectives of the stakeholders of the system and to the quality attributes of the software architecture. Typical stakeholders are customers, business managers, architects and architectural evolution strategists of the organization. Stakeholders have different viewpoints and demand different adaptable content. Adaptability of software architecture is meaningful in specified context, i.e. software architecture is adaptable to specified change requirements.

An objective of the case study has been in validation of Adaptability Evaluation Method (AEM) by means of a real world industrial case. The method has been assisted in requirement engineering, architecture modeling and adaptability evaluation from the architectural models. The case results has been revealed that AEM can be used (1) in requirements engineering, (2) in designing, negotiating and mapping adaptability requirements to the software architectures, and (3) in an adaptability evaluation of the architectures.

The strengths of AEM are largely social. The method assists to focus attention on the important details of the architecture, and allows stakeholders to ignore less critical areas. In the case implementation, the most important source for defining adaptability-related requirements has been provided by the interviews of the development staff of the industrial partner. Weekly meetings have been provided as a successful means of clarifying the meaning of the adaptability-related requirements and achieving a common understanding among the development staff of the target system. Total of six weeks have been used for gathering, defining, and evaluating the adaptability-related issues of the target system.

In the case implementation, adaptability evaluation has been performed with a scenario-based analysis at the architecture level. The exploitation of scenarios has been proven to be an important tool for both communication among a team of developers and for communication between a development team and upper-level managers. Architectural descriptions have been played an important role in scenario-based evaluation. Therefore, it has been important that architectural descriptions have been designed to serve quality evaluation, thanks to the various viewpoints of QADA® that are especially designed for quality evaluation. Each of the viewpoints focuses on specific quality attributes. In the case implementation, the most important viewpoints for adaptability evaluation have been concrete structural and concrete deployment. In adaptability degree comparing, the architectures to be evaluated must be designed for same system by means of different architectural styles, or the results are meaningless. The more detailed the description of the architecture, the more accurate is its evaluation results. The required adaptability for the target system has been achieved.

The industrial partner has participated in the case with developing process of the environment control system. An objective of them has been to develop new adaptable software architecture for multiple platforms, based on existing version of the product. Furthermore, other objectives of them have been to advance and to improve their current developing process related to architectures of adaptable software system, and find out more outlined and more controlled way to develop software system architectures. Furthermore, case objectives of them have also been to deliver research material for purposes of the case and to discover professional view to analyze the software architecture of the target system. To keep observed material near by practice, the industrial partner has decided to design new version of the target system hand in hand with the case implementation. After analyzing earlier version of the product, and specifying the change requirements for new version of the target system, the industrial partner has discovered a lot of proposals to resolve architectural problems especially in network level. As a result of the case, the industrial partner has received architectural design documents of the target system on the component level, and networking components even on class level, both depicted by UML2 notation. From project management point of view, the industrial partner has discovered also new usable methods to estimate needed working hours to implement a single part of a system or even a system as a whole. After all, the industrial partner has much been chuffed about the results of the case: they have been able to help research partner in practical case work and they have achieved the objectives they have set for the case. In the case, two new usable and adaptable software architectures for the target system for multiple platforms have been developed. In addition, improvements for current developing process related to the adaptable software architectures of the industrial partner have been provided by following the phases and steps of AEM.

FUTURE WORK

We will continue the research work related to adaptability issues at architecture level. The next step of this work is to validate AEM with other concrete industrial cases, i.e. with the cases focusing on adaptability issues of Service Architectures, Product Line and Product Family Architectures. Validation of the method will emphasize the special characteristics of these kinds of architectures.

In the case implementation, adaptability degree has been calculated by means of Excel sheets. However, better tool support is needed to facilitate this work. We aim to develop a tool for this purpose in the near future. For example, the tool would calculate adaptability degree of software architecture directly from the architecture diagrams developed by the UML2 notation. Furthermore, the tool would have connection to the stylebase to facilitate software architect in selection of different architectural styles and patterns related to adaptability.

ACKNOWLEDGEMENTS

This research work is mainly conducted in the MERLIN project under the ITEA cluster projects of the EUREKA network and financially supported by Tekes (the National Technology Agency of Finland). Furthermore, the work is partially conducted in the EU funded SMEPP project.

REFERENCES

[1] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*. Massachusetts: Addison-Wesley, 2003, pp. 512.

[2] M. Matinlassi, and E. Niemelä, ”The impact of maintainability on component-based software systems”, in Proceedings of the 29th EUROMICRO Conference (EUROMICRO’03), ”New Waves in System Architecture”, 2003, pp. 25-32.

[3] R. L. Glass, *Software Conflict - Essays on the Art and Science of Software Engineering.* New Jersey: Yourdon, 1991, pp. 367.

[4] ISO/IEC Std. 9126-1, ”Software engineering - product quality - part 1: Quality model”, International Organization for Standardization / International Electrotechnical Commission, Tech. Rep. TR 9126-1:2001(E), 2001.

[5] L. Chung, B. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering.* Boston, Dordrecht: Kluwer Academic Publishers, 1999, pp. 476.

[6] K. Henttonen, M. Matinlassi, E. Niemelä, and T. Kanstrén, ”Integrability and Extensibility Evaluation from Software Architectural Models – A Case Study”, *Open Software Eng. J.*, vol. 1, pp. 1-20, 2007.

[7] D. Garlan, ”Software architecture: A roadmap”, in *The Future of Software Engineering* A. Finkelstein, Ed. New York: ACM Press, 2000, pp. 93-101.

[8] IEEE Std. 1471-2000, ”Recommended Practice for Architectural Description of Software-Intensive Systems”, *IEEE, Institute of Electrical and Electronics Engineers Inc.*, pp. 29, 2000.

[9] K. Smolander, ”Four metaphors of architecture in software organizations: Finding out the meaning of architecture in practice”, in Empirical Software Engineering International Symposium, 2002

[10] N. Rozanski, and E. Woods, *Systems Architecture: Working with Stakeholders using Viewpoints and Perspectives.* New York: Addison-Wesley Professional, 2005, pp. 546.

[11] van der Raadt, B., J. Soetendal, M. Perdeck, and H. van Vliet, ”Polyphony in architecture”, in The 26th International Conference on Software Engineering, 2004

[12] P. Clements, R. Kazman, M. Klein, D. Devesh, S. Reddy, and P. Verma, ”The duties, skills, and knowledge of software architects”, in The Working IEEE/IFIP Conference on Software Architecture (WICSA), 2007, pp. 20.

[13] P. Tarvainen, ”An approach to evaluate the adaptability of software architectures”, in Proceedings of the 5th Workshop on System Testing and Validation (STV 2007), Held in Conjunction with the 20th International Conference on Software & Systems Engineering and their Applications (ICSSEA 2007), 2007, pp. 9-21.

[14] P. Tarvainen, ”Adaptability evaluation of software architectures; A case study”, in Proceedings of the First IEEE International Workshop on Software Engineering for Adaptive Software Systems (SEASS 2007), Held in Conjunction with the 31st Annual IEEE International Computer Software and Applications Conference (COMPSAC 2007), Volume 2, 2007, pp. 579-584.

[15] M. Matinlassi, E. Niemelä, and L. Dobrica, Quality-Driven Architecture Design and Quality Analysis Method, A Revolutionary Initiation Approach to a Product Line Architecture. Espoo: VTT Publication 456, VTT Technical Research Centre of Finland, 2002, pp. 138.

[16] A. Immonen, ”A method for predicting reliability and availability at the architectural level”, in *Research Issues in Software Product-Lines - Engineering and Management* T. Käkölä and J. C. Dueñas, Eds. Heidelberg: Springer-Verlag, 2006, pp. 391-446.

[17] L. Davis, R. F. Gamble, and J. Payton, ”The Impact of Component Architectures on Interoperability”, *J. Syst. Software*, vol. 61, no.1, pp. 31-45, 2002.

[18] A. Egyed, N. Medvidovic, and C. Gacek, ”Component based perspective on software mismatch detection and resolution”, in The IEE Proceedings Software, 2000, pp. 225-236.

[19] D. Batory, C. Johnson, B. MacDonald, and D. von Heeder, ”Achieving Extensibility through Product Lines and Domain Specific Languages: A Case Study”, *ACM Trans. Software Eng.*, vol. 11, no.2, pp. 191-214, 2002.

[20] D. Garlan, R. Allen, and J. Ockerbloom, ”Architectural mismatch or why it is so hard to build systems out of existing parts”, in The 17th International Conference on Software Engineering, 1995

[21] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carriere, ”The architecture trade-off analysis method”, in The 4th IEEE International Conference on Engineering of Complex Computer Systems, 1998

[22] L. Dobrica, and E. Niemelä, ”A Survey on Software Architecture Analysis Methods”, *IEEE Trans. Software Eng.*, vol. 28, no.7, pp. 638-653, 2002.

[23] R. Kazman, M. Klein, and P. Clements, ”ATAM^{SM}: Method for architecture evaluation”, Carnegie Mellon University, Software Engineering Institute, Tech. Rep. CMU/SEI-2000-TR-004 ESC-TR-2000-004, 2000.

[24] S. M. Yacoub, and H. H. Ammar, ”A Methodology for Architecture-Level Reliability Risk Analysis”, *IEEE Trans.Software Eng.*, vol. 28, no.6, pp. 529-547, 2002.

[25] P. Bengtsson, N. Lassing, J. Bosch, and H. Van Vliet, ”Architecture-Level Modifiability Analysis (ALMA)”, *J. Syst. Software*, vol. 69, no.1-2, pp. 129-147, 2004.

[26] A. C. Meng, ”On evaluating self-adaptive software”, in Proceedings of the 1st International Workshop on Self-Adaptive Software (IWSAS 2000), 2000, pp. 65-74.

[27] L. Chung and N. Subramanian, ”Process-oriented metrics for software architecture adaptability”, in Proceedings of the IEEE International Conference on Requirements Engineering, 2001, pp. 310-311.

[28] Xia Liu, and Qing Wang, ”Study on application of a quantitative evaluation approach for software architecture adaptability”, in Proceedings of the 5th International Conference on Quality Software (QSIC 2005), 2005, pp. 265-272.

[29] N. Subramanian, and L. Chung, ”Metrics for software adaptability”, in Proceedings of the International Conference on Software Quality Management (SQM 2001), 2001, pp. 95-108.

[30] R. de Lemos, ”A co-operative object-oriented architecture for adaptive systems”, in Proceedings of the 7th IEEE International Conference and Workshop on Engineering of Computer Based Systems (ECBS 2000), 2000, pp. 120-128.

[31] M. Aksit, and Z. Choukair, ”Dynamic, adaptive and reconfigurable systems overview and prospective vision”, in Proceedings of the 23rd International Conference on Distributed Computing Systems Workshops, 2003, pp. 84-89.

[32] J. M. Cobleigh, B. S. Lerner, L. J. Osterwell, and A. Wise, ”Containment units: A hierarchically composable architecture for adaptive systems”, in Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering, 2002, pp. 159-165.

[33] M. Roḿan, F. Kon, and R. H. Campbell, ”Reflective Middleware: From Your Desk to Your Hand”, IEEE Distributed Systems Online (Special Issue on Reflective Middleware), vol. 2, no.5, pp. 13, 2001.

[34] Y. Zhang, A. Liu, and W. Qu, ”Software architecture design of an autonomic system”, in Proceedings of the 5th Australasian Workshop on Software and System Architectures (AWSA 2004), 2004, pp. 5-11.

[35] L. Andrade, and J. L. Fiadeiro, ”An architectural approach to auto-adaptive systems”, in Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS), 2002, pp. 439-444.

[36] K. Herrmann, ”Meshmd1- a middleware for self-organization in ad hoc networks”, in Proceedings of the 23rd International Conference, Distributed Computing Systems Workshops, 2003, pp. 446-451.

[37] P. K. McKinley, E. P. Kasten, S. M. Sadjadi, and Z. Zhou, ”Realizing multi-dimensional software adaptation”, in Proceedings of the ACM Workshop on Self-Healing, Adaptive and Self-MANaged Systems (SHAMAN), Held in Conjunction with the 16th Annual ACM International Conference on Supercomputing, 2002, pp. 8.

[38] M. Agarwal, V. Bhat, H. Liu, V. Matossian, V. Putty, C. Schmidt, G. Zhang, L. Zhen, M. Parashar, B. Khargharia, and S. Hariri, ”Automate: Enabling autonomic applications on the grid”, in Proceedings of the 5th Annual International Workshop on Autonomic Computing Workshop, Active Middleware Services, 2003, pp. 48-57.

[39] J. Bosch, and P. Molin, ”Software architecture design: Evaluation and transformation”, in Proceedings of the IEEE Engineering of Computer Based Systems Symposium (ECBS'99), 1999, pp. 4-10.

[40] N. Subramanian, and L. Chung, ”Architecture-driven embedded systems adaptation for supporting vocabulary evolution”, in Proceedings of the International Symposium on Principles of Software Evolution, 2000, pp. 144-153.

[41] IEEE Std. 610.12-1990, ”Standard Glossary of Software Engineering Terminology”, *IEEE, Inst. Elect. Electron. Eng. Inc.*, pp. 84, 1990.

[42] P. Oreizy, M. M. Gorlick, R. N. Taylor, D. Heimbigner, G. Johnson, N. Medvidovc, A. Quilici, D. S. Rosenblum, and A. L. Wolf, ”An Architecture-based Approach to Self-adaptive Software”, *IEEE Intell. Syst. Appl.,* vol. 14, no.3, pp. 54-62, 1999.

[43] L. Chung, D. Gross, and E. Yu, ”Architectural design to meet stakeholders requirements”, in Proceedings of the TC2 First Working IFIP Conference on Software Architecture (WICSA1), 1999, pp. 545-564.

[44] P. Grünbacher, A. Egyed, and N. Medvidovic, ”Reconciling Software Requirements and Architectures with Intermediate Models”, *Software Syst. Model.*, vol. 3, no.3, pp. 235-253, 2004.

[45] OMG, *Unified Modeling Language: Superstructure Version 2.0.* Needham, MA, U.S.A.: Object Management Group, 2005, pp. 710.

[46] E. Niemelä, J. Kalaoja, and P. Lago, ”Toward an Architectural Knowledge Base for Wireless Service Engineering”, *IEEE Transactions on Software Eng.,* vol. 31, no.5, pp. 361-379, 2005.

[47] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerland, and M. Stal, *Pattern Orient Software Arch. A System of Patterns,* vol. 1, Chichester: John Wiley & Sons, 1996, pp. 476.

[48] D. C. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects.*, vol. 2, New York: Wiley & Sons, 2000, pp. 666.

[49] T. T. Do, M. Kolp, and A. Pirotte, ”Social patterns for designing multi-agent systems”, in Proceedings of the 15th International Conference on Software Engineering and Knowledge Engineering (SEKE'03), 2003, pp. 103-110.

[50] J. Bosch, *Design and use of Software Architectures: Adopting and Evolving a Product Line Approach.* Harlow: Addison-Wesley, 2000, pp. 368.

[51] M. Shaw, and D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline, 1st Ed.* New Jersey: Prentice Hall, 1996, pp. 242.

[52] B. P. Douglass, *Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns.* Boston, MA: Addison-Wesley Professional, 1999, pp. 800.

[53] D. Pakkala, P. Pääkkönen, and M. Sihvonen, ”A generic communication middleware architecture for distributed application and service messaging”, in Proceedings of the Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services (ICAS/ICNS 2005), 2005, pp. 22-30.

[54] WWW-pages of Object Management Group (OMG). (2008, Mar. 16). CORBA component model, v.4.0. Available: http://www.omg.org/technology/documents/formal/ components.htm

[55] S. W. Cheng, D. Garlan, B. Schmerl, P. Steenkiste, and N. Hu, ”Software architecture-based adaptation for grid computing”, in Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11 2002), 2002, pp. 389-398.

[56] L. Dobrica, and E. Niemelä, ”Adaptive middleware services”, in Proceedings of the IASTED’02 International Conference, Applied Informatics (AI), 2002, pp. 137-142.

[57] D. Garlan, S. W. Cheng, and B. Schmerl, ”Increasing system dependability through architecture-based self-repair”, in *Architecting Dependable Systems* , vol. 2677 of LNCS, R. d. Lemos, C. Gacek and A. Romanovsky, Eds. Berlin, Heidelberg: Springer-Verlag, 2003, pp. 33-38.

[58] WWW-pages of Sun Microsystems. (2008, Mar. 16). Desktop java, java beans. Available: http://java.sun.com/ products/javabeans/

[59] D. O. Keck, and P. J. Kuehn, ”The Feature and Service Interaction Problem in Telecommunications Systems: A Survey”, *IEEE Trans. Software Eng.,* vol. 24, no.10, pp. 779-796, 1998.

[60] Z. Morley, M. Eric, A. Brewer, and R. H. Katz, ”Fault-tolerant, scalable, wide-area internet service composition”, Berkeley, California, University of California, Computer Science Division, Tech. Rep. UCB/CSD-1-1129, 2001.

[61] WWW-pages of Sun Developer Network (SDN). (2008, Mar. 16). The Java ME platform. Available: http://java.sun.com/javame/index.jsp

[62] M. A. Babar, and I. Gorton, ”Comparison of scenario-based software architecture evaluation methods”, in Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC), 2004, pp. 600-607.

[63] G. Costagliola, F. Ferrucci, G. Tortora, and G. Vitiello, ”Class Point: An Approach for the Size Estimation of Object-Oriented Systems”, *IEEE Trans. Software Eng.*, vol. 31, no.1, pp. 52-74, 2005.

Received: March 14, 2008 Revised: April 30, 2008 Accepted: April 30, 2008