Martin Hoffmann

Quality Evaluation of Software Architecture with Application to OpenH.323 Protocol

Master's Thesis
in Information Technology
3rd October 2006

University of Jyväskylä

Department of Mathematical Information Technology

Jyväskylä

Author: Martin Hoffmann

Contact information: marhoffm@cc.jyu.fi, martin.hoffmann@student.hpi.uni-potsdam.de

Title: Quality Evaluation of Software Architecture with Application to OpenH.323 Protocol

Project: Master's Thesis in Information Technology

Page count: 88

Keywords: Quality Control, Quality Attributes, Metrics, Software Architecture Evaluation, ATAM, Telecommunication, H.323

Eigenständigkeitserklärung

Ich versichere hiermit, diese Arbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfsmittel bedient zu haben.

Martin Hoffmann Potsdam, den 03.10.2006

Abstract

The requirements towards software systems usually go beyond the correct functionality, the presence of certain quality demands are also very essential for the systems' acceptance by the stakeholders. So quality control and management must be carried out through the whole development process to ensure the implementation of required quality characteristics. This thesis focuses on the quality control of the software architecture. Several approaches for evaluating the architecture are presented. Furthermore the OpenH.323 protocol architecture is evaluated in a case study. That software architecture is evaluated with two approaches: Architecture Trade-Off Analysis Methode (ATAM) and architectural metrics.

Zusammenfassung

Die Anforderungen an Softwaresysteme gehen weit über die reine Funktionalität hinaus. Für die Akzeptanz bei den Stakeholdern eines Softwaresystem sorgen vor allem Qualitätseigenschaften wie Benutzbarkeit, Effiziens, Sicherheit und Robustheit. Um die Realisierung dieser Eigenschaften des Systems zu gewährleisten, müssen Qualitätskontrolle und -management während des gesamten Entwicklungsprozesses durchgeführt werden. Diese Arbeit untersucht in diesem Zusammenhang die Möglichkeiten der Qualitätskontrolle der Software-Architektur. Es werden verschiedene Ansätze vorgestellt und darüber hinaus wird die Sotware-Architektur des OpenH.323 Protokolls hinsichtlich der Erfüllung von Qualitätseigenschaften bewertet. Zur Evaluierung werden die Methodiken *Architecture Trade-Off Analysis* Methode (ATAM) und *Architekturmetriken* genutzt.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Overall Context of the Research Area

Nowadays, software systems are utilized in many industrial and service-oriented fields either as part of products or for supporting the production. The tasks performed by these systems are very different. For instance, software systems are responsible for controlling critical processes in product of the automobile and aerospace industry, or are important parts of information and communication systems (ICT-systems). These ICT-systems, for instance, support companies' business processes.

Since software performs such important tasks, it is essential for the systems' stakeholders and also for the systems' environments (e.g. other software and hardware systems) that software systems fulfil their required functionality correctly. Even if those systems provide the required functionality it nevertheless is possible that the system's stakeholders are not satisfied with the system because of the absence of so-called non-functional requirements or quality characteristics like, for instance, safety, performance, reliability, availability, or maintainability. Furthermore, the increasing complexity of the software systems which also consist of third-party or common-of-the-shelf (COTS) software components, complicates the development of software systems satisfying the expectations of all stakeholders. To ensure that a system disposes beyond its required functionality the quality control of these non-functional requirements is necessary. Usually, the quality of the system is evaluated regarding the functional and non-functional requirements by testing. But testing the implemented solution is performed at a relatively late stage in the development process and found insufficiencies regarding the system's requirements have to be corrected in the previous development phases. Nevertheless, insufficiencies are sometimes recognized after the system is installed and running, for example, during the maintenance. To enforce changes at that point in the system's life cycle is even more cost intensive. This means if it would be possible to detect at least some of those insufficiencies in earlier development stages would be less cost and resource intensive. The first important output of the development process is the architecture which is a system description regarding the system's structure and behaviour as described in Chapter 2. That means an evaluation of the architecture regarding the functional and non-functional requirements could serve detecting architectural flaws which lead to mentioned system's weaknesses. Corrections in early development stage are much easier to implement than in the later stages of the development process. In general, the early identification and estimation of risks related to architectural decisions would improve the system's quality and should be part of the development process. That is why architecture evaluation has been a growing area of interest in academia and ICT-industry lately. The research project AISA ([5]) addresses the investigation of architecture evaluation. Actually, the research topic of the AISA project is the quality management of enterprise and software architectures in the development of organizations and information systems, as well as strategies, methods, and tools for it. The aim of quality management activities related to the software architecture is to create and maintain a software architecture that enables the system to attain its desired quality attributes. Architectural key success factors, evaluation criteria, and metrics both at enterprise and software architecture level are also investigated in the AISA project. Furthermore, this project aims on research and development of quality management strategies and methods for architectures, particularly evaluation methods. This thesis is related to the investigation of evaluation criteria and possibilities for software architecture evaluation.

1.2 Research Problems and Questions

The general research problem question is how well can one control the quality of software architecture by analyzing the architectural description. This means which methods and measurements can be utilized to gain the relevant information for assessing the system's quality attributes from the architectural description and what are the required descriptions to perform such an evaluation. Actually, in the context of this paper the Architecture Trade-Off Analysis Method (ATAM) is used for evaluating the software architecture of OpenH.323 protocol. The related matters of interest are:

- What kind of architectural description is needed?
- How can it be used with ATAM?
- Is possible to evaluate also runtime characteristics like efficiency?

Since the architecture is the system's structural and behavioural description, it in fact basically permits or excludes a system to fulfil its quality attributes. That circumstance should enable the evaluation of such a software architecture regarding quality attributes and quality metrics by analyzing the architecture's design. Some quality attributes are performance, security, and reliability as well as functionality and extensibility.

These attributes may conflict, and trade-offs among alternative design decisions are an essential part of designing a software architecture. According to [1], at the moment software architecture evaluation criteria and metrics are neither established nor detailed like in contrast the quality criteria of a software product which are even standardized. Further, it is not specified what kind of metrics are measurable by design analysis to make statements on quality.

1.3 Objective of this Work

This thesis aims on the evaluation of the OpenH.323 protocol's software architecture. This Open Source protocol software enables video conferencing over IP networks. The evaluation is performed with the Architecture Trade-Off Analysis Method (ATAM). The ATAM reveals how well the architecture satisfies particular quality goals and since it recognizes that architectural decisions affect more than one quality attribute that means this method enables the identification of trade-offs among several quality attributes. ATAM is described in Section 4.3.3. To perform the ATAM evaluation, the main stakeholders and their quality requirements are identified. Furthermore the architectural description for the evaluation is elaborated from the source code. The architecture must be described from different points of view to address different stakeholder roles as described in Section 3.2. The different description views are described in Section 2.1.

1.4 Structure of the Thesis

The second chapter of this paper defines the term *software architecture* and presents different views on software architecture which have to be considered by an architectural description to address the different stakeholders and aspects of software architecture. *Krutchen's 4+1 views* ([9]) are introduced and discussed. Furthermore, it is stated why for the case study *Soni's conceptual view* ([10]) is utilized. Also the description techniques which are used to illustrate the views are introduced.

The third chapter deals with software quality. The quality of a software system is primarily perceived by its stakeholders because they have the best impression if the software system meets their requirements towards it. These requirements are based on the stakeholders' expectations and needs. Usually quality models are utilized to evaluate the quality of a software system. These models map quality characteristics like efficiency, maintainability, security, and usability to measures. Further, an approach of a role-base quality model is given, this model also considers the relation between stakeholder roles and quality characteristics. Furthermore, the mentioned quality may conflict, and trade-offs among alternative design decisions are an essential part of designing a software architecture.

The fourth chapter deals with the evaluation of the software system's quality characteristics on the architectural level. Software architecture evaluation is the assessment of a software architecture regarding stakeholders' requirements which includes the system's functionality and its quality attributes. In this chapter it is described which methods and approaches exist for evaluating the software architecture regarding required quality characteristics.

The fifth chapter introduces the OpenH.323 system. Firstly, the system's main components are described. Then the protocol is presented with its subprotocols and their tasks. Furthermore, two usage scenarios of the protocol are given to illustrate the H.323 protocol's functionality. Finally the software architecture of the protocol is described. Therefore, the conceptual and logical view as well as the runtime behaviour of the protocol's software components are described.

In the sixth chapter the example evaluation is performed by using a scenario-based method and the architectural metrics approach. The evaluation outputs are presented and analysed. This chapter is followed by a general conclusion and discussion.

2 Software Architecture

2.1 Meaning of Software Architecture

Since this paper deals with the evaluation of software architectures it is necessary to define what the term *software architecture* means in the context of this paper. There is no single, universally definition, but many definitions [2, 3, 7] define software architecture in a similar manner. The *software architecture* basically must describe the software system's components. That means their structure as well as their behaviour and interaction with each other because the whole software system's behaviour results from its components' behaviour. The authors of [2] define software architecture as follows: The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

2.2 Architectural views and descriptions

Most literary sources [2, 9, 10] agree in the point that for a software architecture description different views are necessary for describing the different aspects of a software system. Among these views are description of the static organization of the software system and the development environment as well as the description of the hardware architecture.

A well-known approach is Kruchten's model of the *4+1 views* [9]. This model consists of five main views:

- logical view
- process view
- development view
- physical view
- scenarios

In the following these views are explained: The *logical view* is an object-oriented decomposition of the software system. This view mainly supports the system's functional requirements because the decomposition in *objects* is achieved by abstracting from these functional requirements. That means a certain object or group of objects fulfils a certain functional requirement. These objects exploit the principles of abstraction, encapsulation, and inheritance. The means of description of the logical view are mainly Unified Modelling Language (UML) representations like *object* and *class diagrams* which contain information on the objects' or classes' operations and characteristics. Similar views also exist in other architectural description models, e.g. in [2] this view is called module view because instead of objects the system is decomposed into modules which fulfil certain functionality. Although this view supports the fulfilment of functional requirements it does not describe the system under execution which means the runtime behaviour is not described. Therefore, Krutchen's model also includes the process view. The *process view* considers some non-functional requirements like performance and availability. Furthermore aspects like concurrency, distribution, system's integrity, and fault tolerance are taken into account. Addressing these aspects is supported because the process view considers the execution of *processes* which consist of *tasks*. A task can, for example, be implemented as thread that means a task is the most elementary unit of control. The process view describes the scheduling of tasks and also their distribution over several parallel machines. Also the *inter-task communication* is captured with this view by mechanisms like synchronous and asynchronous message-based communication services, remote procedure calls, event broadcast, rendezvous or shared memory. The logical and process view are connected by the relation between the logical view's objects' operations and the tasks of the process view, because the operations correspond to runtime tasks. In the literature [2] also the terms *component* and *connector* are often used regarding the runtime behaviour description. At this *component* refers to a running system component and *connector* refers to the interaction between two or more of these components. Obviously, a component is comparable with Kruchen's process [9] and a connector with one of the inter-task communiction means used in Kruchten's process view [9].

The *development view* describes the software architecture as an organization of modules and subsystems. The purpose of this decomposition is to create chunks which can be developed concurrently by different development teams. The *physical view* focuses on mapping the software on an underlying hardware platform. It mainly considers which processes of the process view or subsystems of the development view are executed on which hardware node. Finally, the *scenario view* reflects the important functional requirements which the system must fulfil. That is why the scenarios are similar to the use cases in the requirement documents. The four other views have to describe how the fulfilment of the scenarios is supported and implemented by the architecture.

Obviously, Krutchen's 4+1 views [9] aim on support different stakeholder roles by identifying the architecture's aspects they are interested in. That means each view transfers knowledge about the architecture on an abstraction level which addresses only certain stakeholder roles. Systems engineers get the facts which they want to know from the physical view, then the process view. End-users, customers, data specialists obtain their needed information from the logical view. Project managers and software configuration staff use the development view.

I disagree with the author of [9] in the point that end-users and customers can obtain the knowledge about the architecture from the logical view. Since the author suggests the use of UML class diagrams to describe this view, I think that this code-based description is more understandable for developers than for users and customers. Furthermore the decomposition of the architecture into objects which exist at runtime in the memory is probably the wrong means for explaining the structure of system on a general level. That is why I prefer also to use a *conceptual view* for the architectural description. Such a conceptual view is described in Soni's architectural description model in [10]. This conceptual view is a very high-level structure of the system, using functional components and relationships (connectors) between them. The conceptual architecture description is independent of implementation decisions and interaction protocols between components. This ensures that the level of abstraction is general enough that it should be understandable for all groups of stakeholders even for those with a less strong technical background like end-users or customers.

Soni's view model ([10]) proposes three further views which are:

- module architecture
- execution architecture
- code architecture

The module architecture is a decomposition of the system into subsystems, functional modules, and interfaces between them. So implementation details are taken into account. This view corresponds to Krutchen's development view ([9]) which describes the software architecture as an organization of modules and subsystems too. Soni's execution architecture regards the system at runtime. It describes the dynamic structure of the system in terms of tasks, processes, and address spaces. Further the communiction between these mentioned runtime elements and the allocation of resources is described by that view. The execution architecture can be mapped to Krutchen's process view ([9]). According to [10], the code architecture is used to organize the source code into language level modules, directories, files, and libraries. This view is not regarded by Krutchen's model. In contrast to Krutchen's 4+1 views ([9]), Soni's model does not take the underlying hardware platform into account

For the case study, the implementation of the OpenH.323 system used as described in Section 5.1. Knowledge of the structural and behavioural characteristics is gained through reverse engineering. At first the architecture from the conceptual point of view is described to give a general overview of the functional components with their relations towards each other. This description is as far as possible independent of implementation details. So this architectural description belongs to an early development stage and should be suited for an early evaluation method like ATAM which is described in Section 4.3.3. As description means for the conceptual architecture, as described in [10], Fundamental Modeling Concepts (FMC) block diagrams are used, they enable an abstract composational description of the software system. Their main graphical elements are agents, storages and channels. FMC notations are described in [4]. The conceptual description can be elaborated by a description of Kruchten's logical view [9]. That means a description of objects and the functionality they implement. The logical view will be described with UML class diagrams as the author of [9] proposes. Both description illustrate the structure of the software system but they do not contain much information of the system's runtime behaviour. Therefore a description according to Krutchen's process view [9] is necessary. The used means of description are petri nets and UML sequence diagrams.

This paper foregoes the development and physical view because the decomposition into subsystems for enabling concurrent development is not relevant for the case study. A description of the underlying hardware is not necessary either because the OpenH.323 software is only executable on a single processor x86 hardware platform.

The description of the OpenH.323 protocol software architecture is given in Section 5.3.

2.3 Utilization of Software Architecture

Software Architecure described through different views which are introduced in the previous Section, can be utilized for several essential tasks during the development process of the system. The following purposes of software architecture are a summary of software architecture's purposes which can be found in the book [2] and in the paper [6]. First of all, the architecture is a means of communication, which means several stakeholders like users, developers, administrators, managers, and architects can discuss and negotiate about the characteristics of the software system by using the software architecture as a proxy of the planned software system. In addition, the software architecture presents the earliest design decisions, which means the software architecture is basis for further development, deployment, and the maintenance of the software system. Hence, the software architecture has a deep impact on the software system's whole life cycle. Moreover, parts of an architecture can be reused as components for an other architecture. The authors of [2] call this reuse of a software architecture *transferable abstraction of a system.* That means if an architecture contains components which fulfil single and precise tasks and they are relatively independent of other components than those components can be reused in other architectures. That is why not only the reuse aspect should be mentioned, the architecture can also enable the integration of third-party components by using the components' interfaces. This ability to integrate other components is also called *openness* of an architecture. Furthermore, the architecture defines what exactly has to be development so it is a guideline and a means of control for the system development. Then the software architecture can be evaluated regarding the system's fuctional and non-functional requirements. That means the architecture is also used for quality control and quality assurance. Software architecture evaluation is described in Section 4.1.2.

3 Software Quality

3.1 Quality Models

3.1.1 Quality Models in General

Software quality models have been a research matter since the 70's which is shown by the penned literature at that time [11, 12]. In their paper [12] the authors give an overview of quality attributes and create a quality model which mainly corresponds with the ISO 9126-1 quality model (ISO/IEC, 1998) which standardizes a software product's quality, as described in Section 3.1.2.

To ensure that a software product correspond to the demanded software quality so called quality models have been developed. These models are based on the decomposition of the overall quality into abstract quality characteristics which are necessary for meeting the stakeholders' requirements. Each of these characteristics can be further refined into one or more sub-characteristics which also can be refined again. This refinement process results in a tree of quality characteristics whose leaves are concrete quantifiable quality indicators. These quality indicators are so called *metrics*.

There are two classes of software metrics: process metrics and product metrics. The process metrics are used for evaluating the development process; common metrics are cost and time. The second class contains the product metrics which are needed for evaluating the product's quality attributes. In the context of this paper the definition of IEEE Standard 1061 [8] is used. It defines *metric* as a function which assigns to a software unit a value, this value represents the degree of fulfilment of a certain quality attribute. Metrics can be further distinguished in *external* and *internal* metrics. Internal metrics are applied to the software system under construction that means they measure code characteristics, for example, code complexity is an internal metric. External metrics are applied to the executed software system. A typical external metric is the so-called Mean Time To Failure (MTTF) which represents ratio between the overall operating time of the system and the number of failures $MTTF = \frac{operating\ time}{number\ of\ failures}$. MTTF is one of the metrics used for evaluating the attribute Reliability.

The tree structure of quality characteristics naturally differ according to the different quality requirements for different software systems. That is why the existing quality models are actually meta models which have to be adjusted regarding to the specific quality requirements. According to [21] a quality model is taxonomy of quality characteristics for specifying and evaluating non-functional requirements.

3.1.2 ISO 9126-1 Quality Model

In 1998 the ISO/IEC published their ISO 9126-1 quality model (ISO/IEC, 1998) which standardizes quality for every software product and so the implementation of quality into software is adapted to this standardization. This quality model defines six characteristics which serve the fulfilment of functional and non-functional requirements. Quality characteristics only refer to non-functional requirements that is why the functionality is not regarded in the context of that paper. The ISO 9126-1 quality model includes functionality but that conflicts with the definition quality characteristics. But as mentioned above, it depends on the specific non-functional requirements of the software which of these characteristics proposed by the ISO 9126-1 quality model are really relevant for the specific software. The proposed five quality characteristics are described in the following:

- reliability
- usability
- efficiency
- maintainability
- portability

In the following the definitions for all these five quality characteristics according to the ISO 9126-1 quality model, as described in [15], are given.

Reliability

Reliability is the capability of the software product to maintain its level of performance under stated conditions for a stated period of time.

Usability

Usability is the capability of the software product to be understood, learned, used and attractive to the user, when used under specified conditions (the effort needed for use).

Efficiency

Efficiency is the capability of the software product to provide appropriate performance, relative to the amount of resources used, under stated conditions [1] (what the software does to fulfil needs).

Maintainability

Maintainability is the capability of the software product to be modified. Modifications may include corrections, improvements or adaptations of the software to changes in the environment and in the requirements and functional specifications (the effort needed to be modified).

Portability

Portability is the capability of the software product to be transferred from one environment to another. The environment may include organizational, hardware or software environment. Each of these quality characteristics can be refined to a set of sub-characteristics.

The six high-level quality characteristics and their sub-characteristics are shown in Figure 3.1. The fact that the sub-characteristic compliance is part of every quality characteristic catches the eye. Compliance means to adhere to standards, conventions or regulations. The presence of the compliance sub-characteristic means that the remaining sub-characteristics within the each of the six quality characteristic are assumed to be fulfilled by the particular standard.

[1] Specified or stated conditions refer to the environment in which the software product is operating and they are usually stated in the product's specification. Such conditions can be, for example, the interaction with other software products and the system's load.

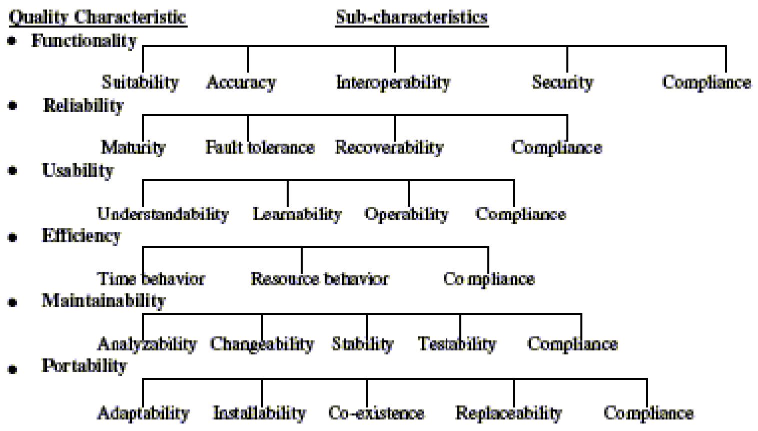


Figure 3.1: ISO 9126-1 quality characteristics with their sub-characteristics [15]

3.2 Stakeholders and Quality Characteristics

The quality of a software system is primarily perceived by its stakeholders because they have the best impression if the software system meets their requirements towards it. These requirements are based on the stakeholders' expectations and needs. Principally, there are two different categories of requirements: functional and non-functional ones. In accordance with [14] functional requirements describe the tasks of the system, whereas non-functional requirements specify overall quality characteristics that means how well does the system fulfils its tasks.

Different stakeholders of a system have different expectation towards the system and so also towards the system's quality characteristics. These different expectation result from different interests or views towards the system. That means the existence or absence of certain quality characteristic is of different meaning for different stakeholders. A single stakeholder is usually not interested in all the quality characteristics of a quality model, e.g. the end user is more interested in characteristics like usability and efficiency, the administrators appreciate maintainability and portability more than other quality characteristics. So the system's quality is the entirety of those quality characteristics which have to be fulfilled to achieve the stakeholders' non-functional requirements. The identification of the stakeholders' requirements is an essential step before designing the software architecture because a not regarded requirement decreases the system quality. To avoid this, the main stakeholder roles and their concerns have to be identified. In [24] the following stakeholder roles are identified:

- user of the system

- customer (client, sponsor, owner)
- component vendor (supplier, contractor)
- analyst
- quality assurance team
- system administrator
- maintainer
- developer
- architect
- project manager

Naturally, these stakeholder roles are a generalisation and cover all possible main stakeholders. That is why not all of these roles exist for every project.

The main stakeholder roles of the OpenH.323 protocol software are the ITU, end user, administrator, developer, and architect. The ITU demands certain quality characteristics of the protocol in its H.323 specification, e.g. exchange of request/response pairs within certain time limits.The end user is interested in a secure communication and also fast system replies so that audio and video data is transferred without recognizable delays. For the administrator resource behaviour and portability are import qualtiy characteristics. The developer wants the protocol software to be quite well maintainable regarding future changes and extensions. So the architect has to implement all the stakeholder requirements into protocol software architecture in a manner that the overall quality is satisfying the stakeholders as good as possible.

The entity-relationship diagram (ERD) in Figure 3.2 shows a role-based quality model that means it shows the main stakeholder roles of the OpenH.323 protocol software assigned to the quality characteristics which they demand. The purpose of quality models is described in Section 3.1.1. The used quality characteristics correspond to those in the ISO 9126-1 model which is described in Section 3.1.2. Starting from the role-based quality model in Figure 3.2 in Section 3.2, three import quality characteristics, efficiency, maintainability, and security can be identified for the OpenH.323 protocol software. Figures 3.3 and 3.4 illustrate the mentioned refinement to metrics

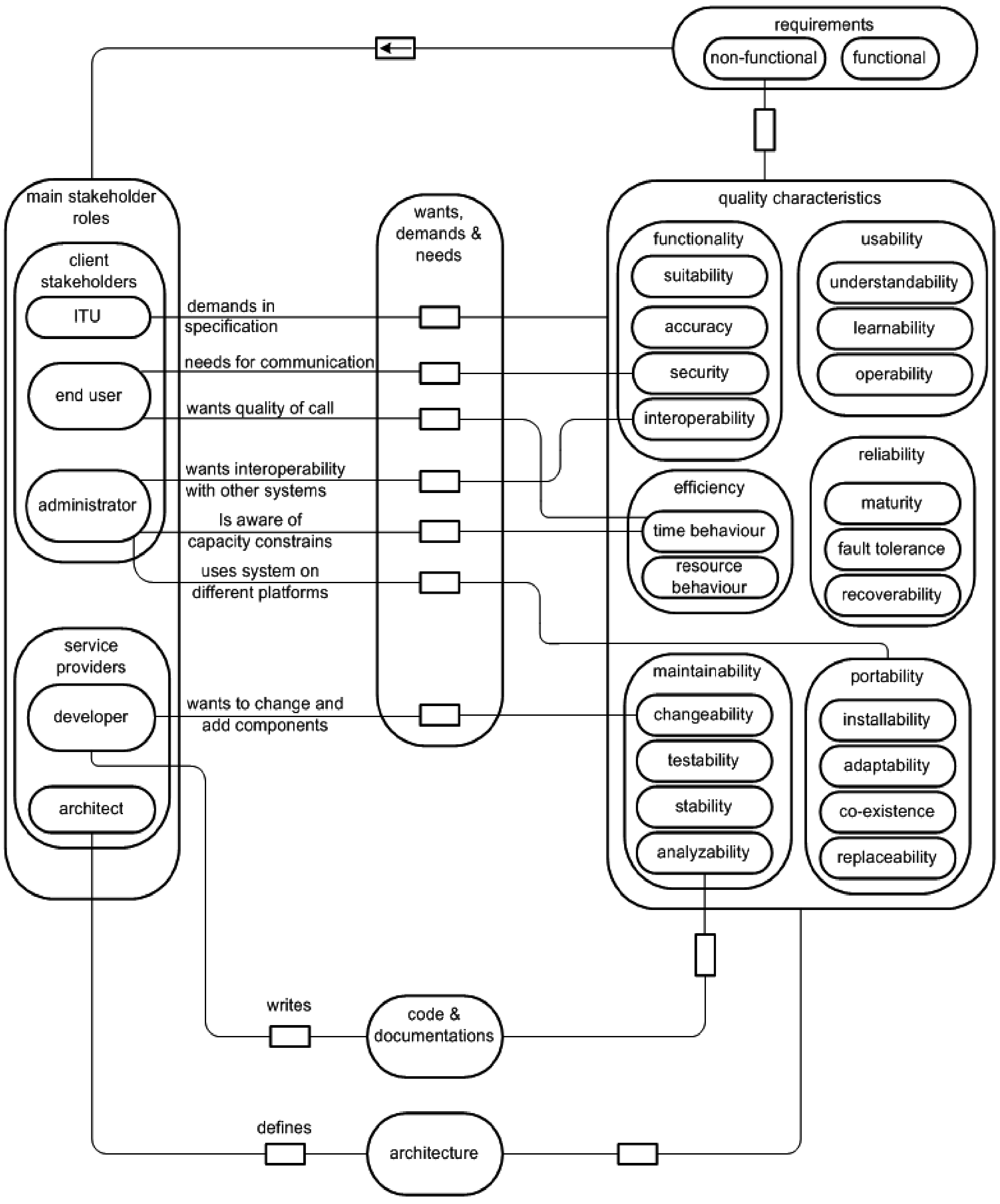


Figure 3.2: Role-based quality model in FMC ER-diagram notation

and quantitive values which enable the evaluation of the three demanded quaracteristics. For the representation, again the FMC ERD notation is used to show relation between characteristic and metrics. On the shown example 3.3 for the refinement of the security characteristic, the quantitave evaluation through measurements is based on the analysis of the utilized authentication, encryption, and auditing algorithms.

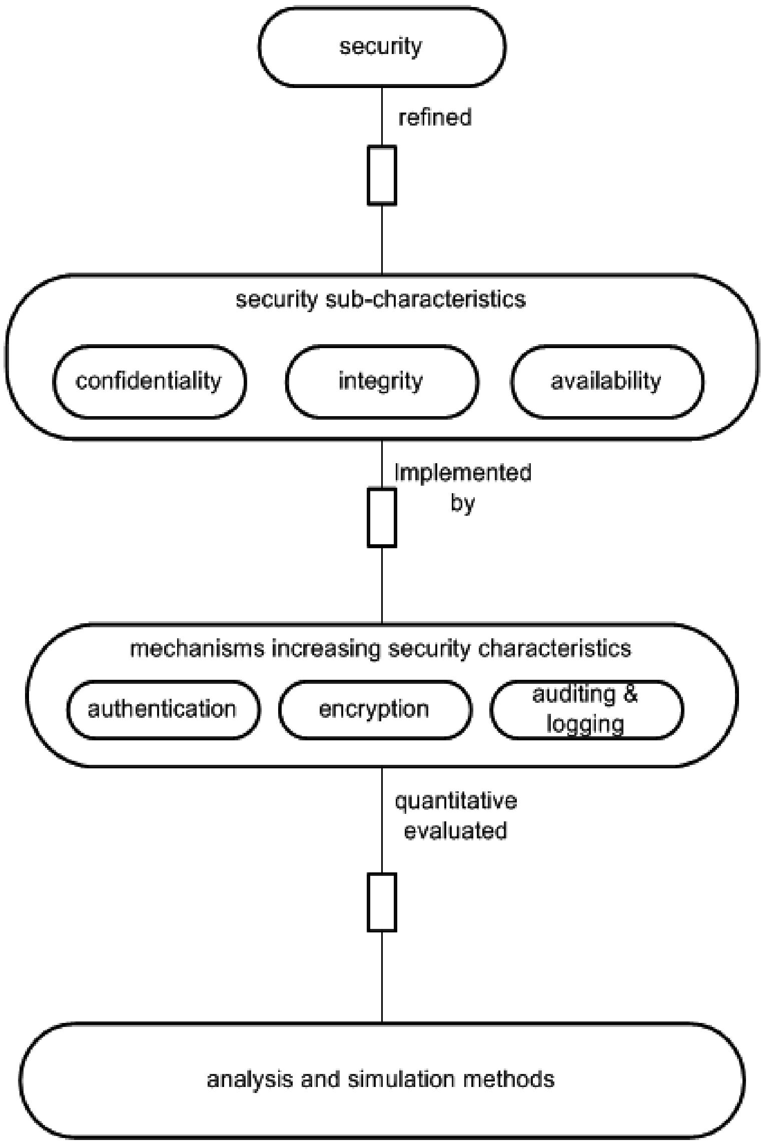
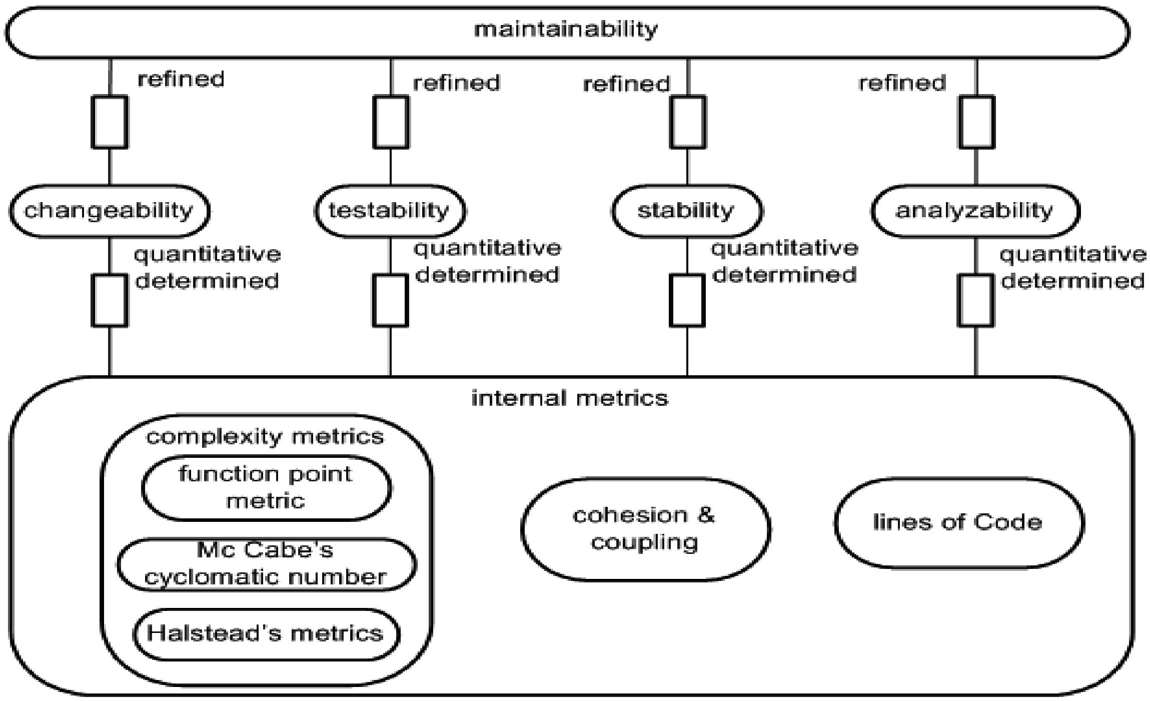


Figure 3.3: Refinement process for quality characteristic security



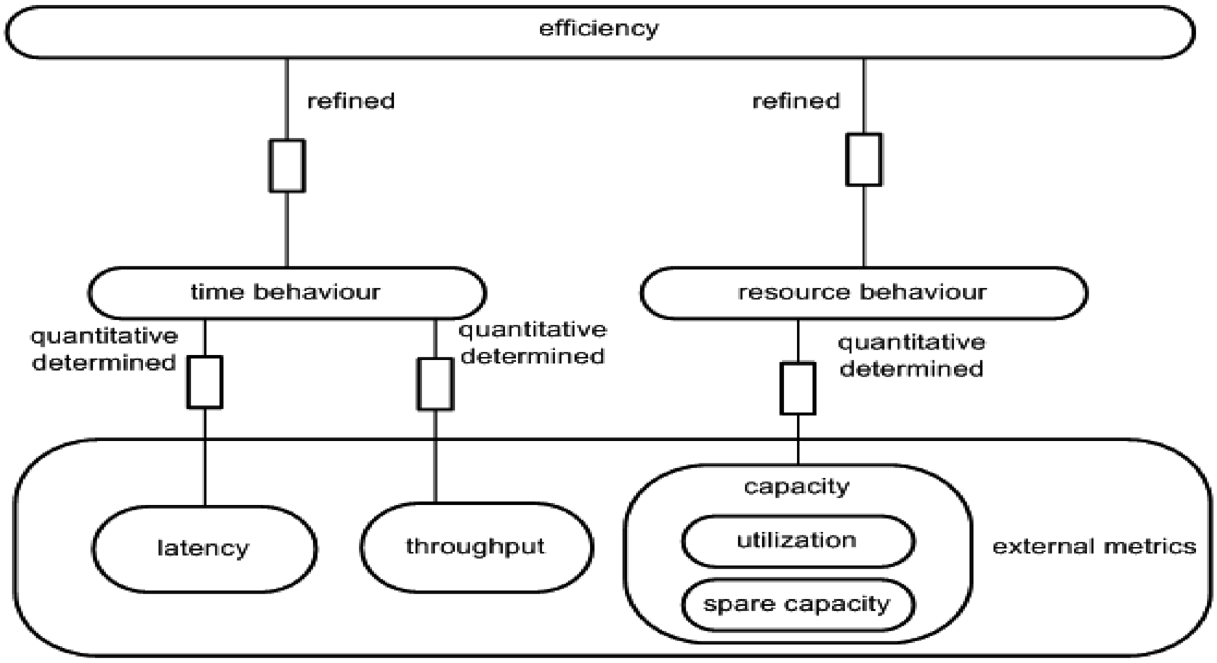


Figure 3.4: Refinement process for quality characteristic maintainance and efficiency

3.3 Software Quality Attribute Trade-offs

The role-based quality model in Figure 3.2 is an approach to relate the ISO 9126-1 quality model to quality requirements of the OpenH.323 protocol's stakeholders. As mentioned above, this model illustrates that the quality characteristics security, efficiency (time and resource behaviour), and maintainability are import characteristics which have to be implemented into the OpenH.323 protocol software.

These quality characteristics among others, like those introduced in Section 3.2, may conflict, and *trade-offs* among alternative design decisions are an essential part of designing a software architecture.

Maintainability is a quite important characteristic for the OpenH.323 protocol software because it should be possible to extend the protocol in case of new subprotocols or data codecs. Since it is an open source software protocol, the structure and understandability of the code and documentations are essential for futher development. Regarding the structuring of the system into different components, there are several design approaches using object-oriented paradigms, design patterns, and component frameworks to improve the maintainability of a system. According to the authors of [27] these techniques often tend to cause a decrease of the system's efficiency because of higher resource consumption, e.g. memory. But for the protocol software also efficiency aspects are essential because the protocol specification defines time limits within which certain protocol request response pairs have to be exchanged. Also the security requirement is influencing the system's performance because implemented encryption algorithms and authentication protocols also need extra time and resources.

Conflicts among quality characteristics must be identified and quantitatively evaluated to find reasonable trade-offs which are necessary to achieve the best possible overall quality for the software system.

Figure 3.5 illustrates the relationship between the single quality characteristics efficiency and maintainability. The relationship is represented by the curve which shows that the increase of one quality characteristic causes the decrease of the other characteristic. So the task of the architect is to find the right balance between the single

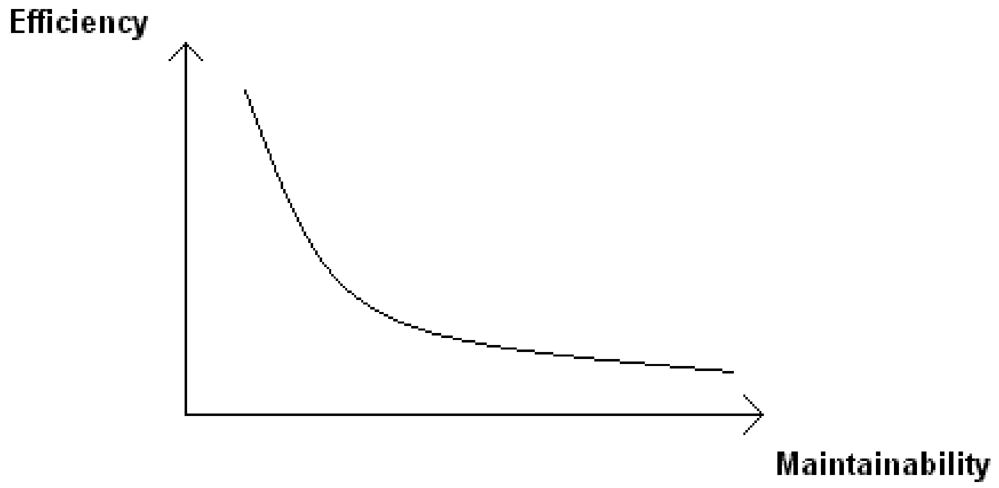


Figure 3.5: Software Quality Attribute Trade-off

quality attributes to achieve a overall quality which satisfies the stakeholders.

A method which supports the identification of trade-offs is the Architecture Trade-Off Analysis Method (ATAM) which is described in Section 4.3.3.

4 Software Architecture Evaluation

In the previous chapters was shown that quality characteristics are related to the stakeholders and that the software architecture must implement those characteristcs. Also the fact that these characteristics might conflict so that trade-offs have to be done between design decisions. *Software architecture evaluation* is the assessment of a software architecture regarding stakeholders' requirements which includes the system's functionality and its quality attributes. In the following, it is described which methods and approaches exist for evaluating the software architecture regarding required quality characteristics.

4.1 Early vs. Late Software Architecture Evaluation

An architecture evaluation can be performed in different stages of architecture creation process. Actually, the authors of [16, 17] distinguish two possible evaluation phases: the *early* and *late evaluation.*

4.1.1 Early Evaluation

Early evaluation is performed when only fragments of the architectural description exist so that mostly the questionnaires (Section 4.3.1), checklists (section 4.3.1), and scenario-based methods (Section 4.3.2) are used for assessment because at this stage there is not enough tangible information available for collecting metrics or simulating behaviour. Mainly the experience of the developers and scenarios based on requirements in the requirement documents are the foundation for the early evaluation.

4.1.2 Late Evaluation

Late architecture evaluation is carried out during later stages of the software development process when there is at least a detailed design available on which more concrete metrics can be collected that means the architectural metrics approach is used to evaluate the software architecture regarding one ore more quality attributes. To ensure the quality control and quality assurance early evaluation and late evaluation techniques should be used in this way. It is possible to ensure that the stakeholders' requirements are considered and implemented in the architecture. This point of view corresponds with [24] because the author proposes first an architectural review which is actually an early evaluation and secondly the determination of relevant quality attributes like architectural metrics (Section 4.3.4), simulation (Section 4.3.5) and mathematical modelling (section 4.3.5). In fact, this second proposition has the same purpose as the late evaluation.

4.2 Goals of Software Architecture Evaluation

According to [16], there are three essential purposes of software architecture evaluation. The first main purpose is the early identification of insufficiencies which were made during the requirements or early design phases. Here insufficiency means that the architecture does not meet the stakeholders' expectation regarding one or more quality attributes. The earlier the development phase in which such an insufficiency is found the less costs and resources are necessary for its elimination. Since the architecture describes the whole software system, unrecognized weaknesses existing in the architecture cause errors in further development outputs like the implementation or in the worst case the final product. Hence architectural changes caused by weaknesses which have been discovered late will also cause necessary changes in these further development outputs. These changes are naturally more resource-intensive than only changes to the architecture itself. The second main purpose is the comparison between alternative architectural suggestions regarding one or more quality attributes. There can be several candidate architectures for the realisation of a software system which implement the same functionality but address quality attributes differently. Then it is necessary to find the most suitable architectural proposal for the realisation of the software system. The third main purpose of the software architecture evaluation is the investigation whether an architecture holds risks for certain quality attributes. The evaluation of software architecture can be seen as a part of another evaluation process. A software system could have been planned with the purpose that it is part of an enterprise architecture because nowadays many business processes of enterprise utilize software for achieving their purposes, for example, through application, telecommunication, workflow, and database software systems. An enterprise architecture also describes a complex and dynamic system which has to fulfil a certain functionality or provide certain services. This system involves much more stakeholders than a software system, and these stakeholders also expect a number of functional and non-function requirements. According to [23] the architectural description of a software system, as well as an enterprise system should regard that the system has to meet functionality, openness, performance, reliability, and maintainability. To create, develop, implement, and maintain an architecture which meets the stakeholders' requirements needs also an evaluation process to enforce quality control and assurance. The three identified main goals of software architecture evaluation are general enough, so that they can also be seen as three targets of enterprise architecture evaluation.

4.3 Software Architecture Evaluation Methods

In the literature [16, 14, 17] five main approaches for software architecture evaluation have been identified:

- Questionnaires and Checklists
- Scenario-based methods
- Architectural Metrics
- Prototyping
- Mathematical Modelling

The first two techniques have a stronger focus on evaluating if the stakeholders' requirements are met by the architecture, and the identification and evaluation of the relevant design decisions implementing these requirements. The last three focus more on evaluating the system regarding the required quality attributes on the basis of measurements and simulations. In the following all approaches are presented.

4.3.1 Questionnaires and Checklist

According to [2], the techniques using questionnaires and checklists are quite similar; both consist of questions regarding the issue if the architecture fulfils functional and non-functional requirements. These questions have to be answered by a group of the software system's stakeholders. That means this evaluation is based on their experience. As well questionnaires as checklists are assessed statistically. An example of a questionnaire-based software architecture evaluation is presented in Svahnberg's paper [17]. In this example, the questionnaire used for this evaluation basically aims on the identified necessary system's quality characteristics. According to these quality characteristics, five quality attributes are investigated on four candidate architectures.

The questionnaire contains four parts: The first part covers generic questions like what architecture (e.g client-server, multi-tier) the participant would prefer based on his/her experience. Moreover, it contains some questions whether there are any architecture types or quality attributes missing. The second part deals with questions to obtain a prioritized list of quality attributes. The third part consists of questions to rate the support given for the quality attributes within each architecture candidate. The fourth part encloses questions to rate which architecture is best at each quality attribute. If possible each stakeholder role of the software system should be involved in the evaluation.

4.3.2 Scenario-based methods

The following explanation is based on the book [16] whose authors developed several scenario-based methods at the Software Engineering Institute, Carnegie Mellon University. Scenario-based techniques evaluate the software architecture by considering it from a higher abstraction level that means the architectural description must neither be complete nor very detailed. A further commonness is that that these methods define a number of steps which have to be performed to achieve a useful evaluation result. These steps are:

- description of the architecture or the architectures which should be evaluated
- development of scenarios (based on non-functional requirements)
- prioritization of the scenarios according to the quality attributes they should prove
- evaluation the architecture from the high-priority scenarios perspectives
- exposition of the results

Scenarios describe the desired system's behaviour during performing certain tasks. This behaviour depends on the existence of certain quality characteristics. That means if the architecture enables the fulfilment of certain scenarios proves the implementation of certain quality characteristics. The quality of the evaluation and especially its results depends on the scenarios' quality. Their quality increases by a well done mapping of requirements to scenarios. It is fatal if an important and necessary scenario is missing during the evaluation. Therefore, the scenario development should involve representatives from all stakeholders.

Such scenario-based methods are for example:

- Software Architecture Analysis Method (SAAM)

- Architectural Trade-off Analysis Method (ATAM)
- Active Reviews for Intermediate Designs (ARID)

All these three example methods were developed at the Software Engineering Institute, Carnegie Mellon University. These methods are described in detail in the book [16].

4.3.3 Architectural Trade-off Analysis Method (ATAM)

In the following the Architecture Tradeoff Analysis Method (ATAM) is described in detail because several steps of it are used for the identification of trade-offs in the case study. The ATAM is so named because it reveals how well an architecture satisfies particular quality goals and since it recognizes that architectural decisions affect more than one quality attribute that means this method enables the identification of trade-offs among several quality attributes. According to [16] the participation of three different groups is usually necessary for performing the ATAM.

- *Evaluation team* is a group of three to five people who are external to the project whose architecture is being evaluated. Each member of the team is assigned a number of specific roles to play during the evaluation. These roles are described in Table 4.1.
- *Project decision makers* are people who are authorized to speak for the development project or have the right to command modifications to it. This group normally consists of the project manager, the customer who is footing the bill for the development, the architect, and the person commissioning the evaluation.
- *Architecture stakeholders* include developers, testers, integrators, maintainers, performance engineers, users, builders of systems interacting with the one under consideration, and others. Their job during an evaluation is to state the specific quality attribute goals that the architecture should meet in order for the system to be considered a success. This group usually consists of twelve to fifteen people.

Role	Responsibilities
Team Leader	Sets up the evaluation; coordinates with client, making sure client's needs are met; establishes evaluation contract; forms evaluation team; sees that final report is produced and delivered (although the writing may be delegated)
Evaluation Leader	Runs evaluation; facilitates elicitation of scenarios; administers scenario selection/prioritization process; facilitates evaluation of scenarios against architecture; facilitates on-site analysis
Scenario Scribe	Writes scenarios on flipchart or whiteboard during scenario elicitation; captures agreed-on wording of each scenario, halting discussion until exact wording is captured
Proceedings Scribe	Captures proceedings in electronic form on laptop or workstation, raw scenarios, issue(s) that motivate each scenario (often lost in the wording of the scenario itself), and resolution of each scenario when applied to architecture(s); also generates a printed list of adopted scenarios for handout to all participants
Timekeeper	Helps evaluation leader stay on schedule; helps control amount of time devoted to each scenario during the evaluation phase
Process Observer	Keeps notes on how evaluation process could be improved or deviated from; usually keeps silent but may make discreet process-based suggestions to the evaluation leader during the evaluation; after evaluation, reports on how the process went and lessons learned for future improvement; also responsible for reporting experience to architecture evaluation team at large
Process Enforcer	Helps evaluation leader remember and carry out the steps of the evaluation method
Questioner	Raise issues of architectural interest that stakeholders may not have thought of

Table 4.1: Evaluation team roles with their responsibilities [16]

The whole ATAM-based evaluation is divided into four phases. The first phase is called *partnership and preparation.* In this phase basically the evaluation team leadership and the key project decision makers informally meet to work out the details of planned evaluation. They agree on formal issues like logistics, such as the time and place of meetings, statement of work or nondisclosure agreements, and then they agree about a preliminary list of stakeholders. Furthermore, they decide which architectural documents will be delivered to the evaluation team for performing the evaluation. The actual *evaluation phases* are the second and third phase. The evaluation team uses the second phase for studying the architecture documentation to get a concrete idea of what the system is about, the overall architectural approaches which are chosen, and the important quality attributes. During the third phase the system's stakeholders join the evaluation team and both groups analyze the architecture together. The analysis is based on the elecitated scenarios. According to [16] the capturing and elicitation of functional and non-functional requirements is part of ATAM. In the fourth and last phase the evaluation team creates and delivers the final report.In the following the concrete steps which are performed during the evaluation are described. The steps one to six belong to the second phase and the steps seven to nine belong to the third phase.

First Step

The first step mainly consists of the presentation of the ATAM with its steps and outputs to the three participating groups mentioned above.

Second Step

During the second step the context for the system and the primary business drivers which are the reasons for the system's development are presented to the involved persons. Business drivers are all the functions, information and people enforcing the business goals of an enterprise and ensuring the daily business. Therefore, the system's most important functions, the enterprise's business goals and their relation to the system, any relevant technical, managerial, economic, or political constraints, and the system's major stakeholders are presented. So actually the desired effect of the system on the its environment is described.

Third Step

In the third step, the architecture is presented at an appropriate level of detail that means the presentation is depending on how much of the architecture has been designed and documented; how much time is available; and the nature of the behavioural and quality requirements. The architectural presentation covers technical constraints like the operating system, hardware, or middleware which are intended to be used, and further it shows other systems with which the system must interact. Most important, the architect describes the architectural approaches used to meet the functional and non-functional requirements. The architecture should be described through different views to address different stakeholder roles, as described in Section 2.2.

Fourth Step

During the fourth step the evaluation team identifies the architectural approaches and used patterns and lists them as a basis for further analysis.

Fifth Step

In the fifth step, the quality attribute goals are formulated in detail using a mechanism known as the utility tree. The evaluation team in cooperation with the project decision makers identify, prioritize, and refine the system's most important quality attribute goals, which are expressed as scenarios. The utility tree serves to make the requirements concrete, forcing the architect and customer representatives to define precisely the relevant quality requirements that they were working to provide.

A utility tree begins with utility as the root node. Utility is an expression of the overall quality of the system. Quality attributes form the second level because these are the components of utility. Typically, performance, modifiability, security, usability, and availability are the children of utility, but participants are free to name their own as long as they are able to explain what they mean through refinement at the next levels. The third level of the utility tree consists specific refinements of the quality attributes, for example, performance might be decomposed into *data latency* and *transaction throughput*. These refinements are the base for the creation of scenarios which form the leaves of the utility tree and they are concrete enough for prioritization and analysis. According to [13], scenarios are the mechanism by which broad and ambiguous statements of desired qualities are made specific and testable. ATAM scenarios consist of three parts:

- *stimulus* which is an event arriving at the system, the event's generator and handler are also named
- *environment* (what is going on at the time)
- *response* (system's reaction to the stimulus expressed in a measurable way)

The definition process of a utility tree is similar to the definition of a quality model for a software product as described in Section 3.1.2 because the overall quality is devided into quality characteristics which are refined in measurable quality attributes which are evaluated by metrics. So metrics are the leaves in a quality model. In the utility tree, scenarios are indicators of certain quality attributes. Of course, a metric is much more concrete because it is a value assigned to an attribute, the scenario in contrast serves to evaluate theoretically wheater it is implemented by the architecture. Some scenarios might express more than one quality attribute and so they might appear in more than one place in the tree. To simplify the analysis, these scenarios should be splitted according to different concerns. The refinement process of quality attributes to scenarios might lead to many scenarios which cannot all be analyzed, so this fifth step also includes the prioritization of the scenarios.

This prioritization can be based on a scale from zero to ten or on a relative ranking like high, low, and medium. The latter one is recommended by [13] because it is less time consuming. The ranking is done by the project decision makers. Furthermore, the scenarios are prioritized by the architect regarding the difficulty of satisfying the scenario by the architecture. There also the high, medium, and low ranking is recommended. Now each scenario has an associated ordered pair (importance of scenario for the system, difficulty of satisfying the scenario by the architecture), for example (H,H). The ordered pair (H,H) means, this scenario is very essential for the system and it is difficult to implement it by the software architecture. The scenarios that are the most important and the most difficult will be the ones where precious analysis time will be spent, and the remainder will be kept as part of the record. A scenario that is considered either unimportant (L,*) or very easy to achieve (*,L) is not likely to receive much attention. The output of utility tree generation is a prioritized list of scenarios that serves as a plan for the remainder of the ATAM evaluation. It tells the ATAM team where to spend its (relatively limited) time and, in particular, where to probe for architectural approaches and risks. The utility tree guides the evaluators toward the architectural approaches for satisfying the high-priority scenarios at its leaves. The utility tree for the ATAM evaluation of the OpenH.323 protocol architecture is shown in Figure 6.1.

Sixth Step

The following sixth step contains of the analysis of the architectural approaches. The architect explains how the high-ranked scenarios are implemented by the architecture and the evaluation team documents the relevant architectural decisions and identifies and catalogues their risks, non-risks, sensitivity points, and tradeoffs. The architect has to explain which approaches and architectural decisions meet the quality requirements.

The upcoming discussion leads to deeper analysis, depending on how the architect responds. The key is to elicit sufficient architectural information to establish some link between the architectural decisions that have been made and the quality attribute requirements that need to be satisfied. At the end of this step, the evaluation team should have a clear picture of the most important aspects of the entire architecture, the rationale for key design decisions, and a list of risks, non-risks, sensitivity points, and trade-off points.

Seventh Step

The seventh step is stakeholder-oriented because the evaluation team asks the group of stakeholders to brainstorm scenarios which are operationally meaningful regarding the stakeholders' individual roles. These scenarios are also prioritized because of the limited time for analysis. First, stakeholders are asked to merge scenarios they feel represent the same behaviour or quality concern. Then they vote for those they feel are most important.

Eighth Step

In the eighth step the architect explains to evaluation team how relevant architectural decisions contribute to realizing each of the chosen scenarios from step seven. During the architect's explanations the evaluation team again identifies and catalogues risk, non-risks, and trade-offs.

Ninth Step

Then, in the ninth step, the gained information from the ATAM needs to be summarized and presented once again to stakeholders.

ATAM's evaluation phase results in the following outputs:

- architectural approaches documented
- set of scenarios and their prioritization from the brainstorming
- utility tree
- risks
- non-risks
- sensitivity points and trade-off points

Finally, the evaluation team groups risks into risk themes. For each risk theme the affected business drivers from the second step are identified. By relating risk themes to business drivers the risk becomes also tangible for non-technical stakeholders like managers.

4.3.4 Architectural Metrics

This approach aims at measuring certain attributes of the software architecture which enable assumptions about the architecture's quality. Architectural metrics belong to the group of product metrics as described in Section 3.1.1. They are derived from quality attributes which are refined quality characteristics. The existing software architectural metrics are quite limited. Furthermore, the so-called architectural metrics are very similar to design metrics. A reason is, according to [16], that the existence of a detailed architectural description is necessary to collect metrics. That means that the design description is at a stage where it can be implemented or parts of it are already implemented. Mostly metrics about structural characteristics are collected. These measurements are performed with the help of the architectural descriptions referring to the different architectural views described in Section 2.2 which are commonly presented in UML notation or on the code level, and so tool-based measurements are also possible. The architectural metrics reflect class characteristics like the complexity of a class, number of methods, depth of the inheritance hierarchy, coupling, and cohesion. The collected metrics are interpreted for evaluating quality attributes especially the attributes maintainability, testability, understandability, reusability, complexity, and also efficiency. *Cohesion* describes the dependencies between methods within a single software component to fulfil a single and precise task. So a high cohesion means that all parts of a component are necessary for fulfilling this task. *Coupling* regards the dependencies between different components. The lower the coupling the more independent are the components from each other and the easier are changes to the system. For many systems an architecture is desired which aims on a maximal cohesion and a minimal coupling. An example of measuring the coupling between modules of software system is given in [18]. Another import metric is the cyclomatic complexity. According to [29], the cyclomatic complexity of a method is the count of the number of paths through the method's source code. Cyclomatic complexity is normally calculated by creating a graph of the source code with each line of source code being a node on the graph and arrows between the nodes showing the execution pathways. A implementations with a high cyclomatic complexity tend to be more error-prone, difficult to test with high coverage, and also more risky regarding maintainability(especially for changeability).

4.3.5 Prototyping

This technique is described in [19, 26] so most of the information given here on Prototyping is based on them. In order to gain information about architectural quality the strategy is similar to the utility tree described in Section 4.3.3. The important quality attributes are refined into scenarios. The necessary functionality to perform these scenarios is implemented in the prototype. The executable prototype can be tested regarding quality attributes at runtime. The gained results are used for further development or correction of the software architecture. The scenarios are mostly implemented without user-oriented and business-oriented aspects of the architecture, what makes the prototyping evaluation approach resource-saving especially regarding time and cost. The prototyping approach is often also called simulation in the literature, e.g. in [24]

4.3.6 Mathematical Modelling

A mathematical model is an abstract model which describes the system's behaviour or certain aspects of the system's behaviour. The model is used for determining theoretically how the system reacts on certain events. According to [24, 25], especially for high-performance computing, reliable systems, real-time systems, etc. mathematical models have been developed, and they can be used to evaluate especially quality attributes related to the runtime behaviour of the system. Different from the other approaches, the mathematical models allow for static evaluation of architectural design models. Mathematical modelling is an alternative to prototyping because both approaches are primarily suitable for assessing runtime behaviour. The approaches can also be combined. Two widely spread types of models are performance modelling and real-time task models. For example, performance modelling can be used to determine the computational requirements of the individual components in the architecture. These theoretical results can then be used and proofed with the running prototype in a simulation. Since the focus of this work also is on the performance assessment with the help of the architecture performance modelling is a suitable approach. Typical performance models are queuing networks, markov chains which are based on stochastic and probability-based methods, and other stochastic approaches like stochastic process algebras.

4.3.7 Summary

While the measurement-based approaches, architectural metrics, and prototyping give concrete values for the evaluation and make it that way a bit more sound, they have the drawback that they can be applied only in the presence of a working artifact. Also the mathematical models are based on detailed description of the whole architecture or at least of some components because the more detailed the model the more realistic are the computed results. Questionnaires and scenario-based evaluation, on the other hand, work just fine on hypothetical architectures, and can be applied much earlier in the life cycle. Actually, these techniques can be seen as architectural review with the main stakeholders because they improve the understanding of the impact of architectural decisions on the system's requirements. Furthermore, even if the architectural description is not in a implemental stage, these approaches are able to identify insufficiencies, weaknesses, and risks.

5 OpenH.323 system

OpenH.323 is an Open Source implementation of the ITU-T H.323 video conferencing protocol. This paper relates to the ITU-T H.323 architecture because OpenH.323 implements this specification and also the developers of OpenH.323 refer to the ITU specification. For the following description of the H.323 architecture, the source [20] is used. The components of H.323 architecture are terminal, gateway, gatekeeper, and multipoint control unit (MCUs). All these components communicate via H.323 protocol.

5.1 H.323 System Components

In the following, the main components of the H.323 system are briefly presented. Figure 5.1 shows the H.323 system components.

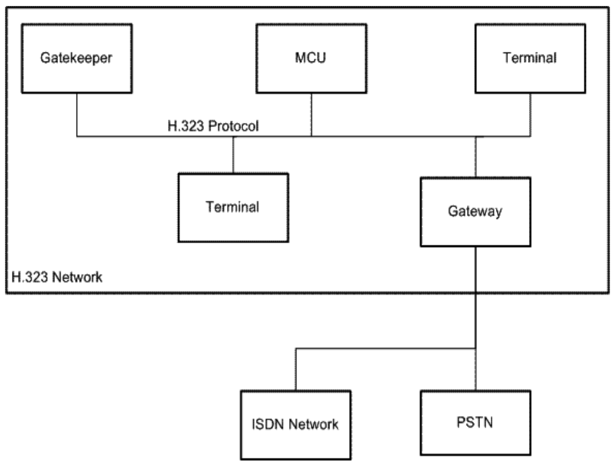


Figure 5.1: H.323 Components

5.1.1 Terminal

The terminal represents the endpoint of every connection. It provides fast two way communications with another H.323 terminal, gateway or multipoint control unit. This communication consists of speech, speech and data, speech and video, or a combination of speech, data and video.

5.1.2 Gateway

The gateway establish the connection between the terminals in the H.323 network and also with terminals belonging to different networks with different protocol stack such as the traditional Public Switching Telephone Network (PSTN) or Integrated Service Digital Network (ISDN).

5.1.3 Gatekeeper

The gatekeeper is an optional component in the H.323 system which is primarily used for admission control and address resolution that means for translating between telephone number and IP addresses. It also manages the bandwidth and provides mechanisms for terminal registration and authentications. Further the Gatekeeper provides facilities such as call transfer and call forwarding.

5.1.4 Multipoint Control Unit (MCU)

The MCU enables establishing and managing multipoint conferences. It consists of a mandatory Multipoint Controller (MC), which manages call signalling and conference control. The second MCU component is a Multipoint Processor (MP), which handles switching and mixing of media stream.

5.2 H.323 Protocol

5.2.1 H.323 Protocol and Subprotocols

The H.323 protocol is a collection of several protocols which enable the transfer of voice and video data over a network. This protocol is also used for teleconferencing via Public Switched Telephone Network (PSTN) and Integrated Services Digital Network (ISDN). The protocols which are utilized by the H.323 protocol architecture are shown in Figure 5.2.

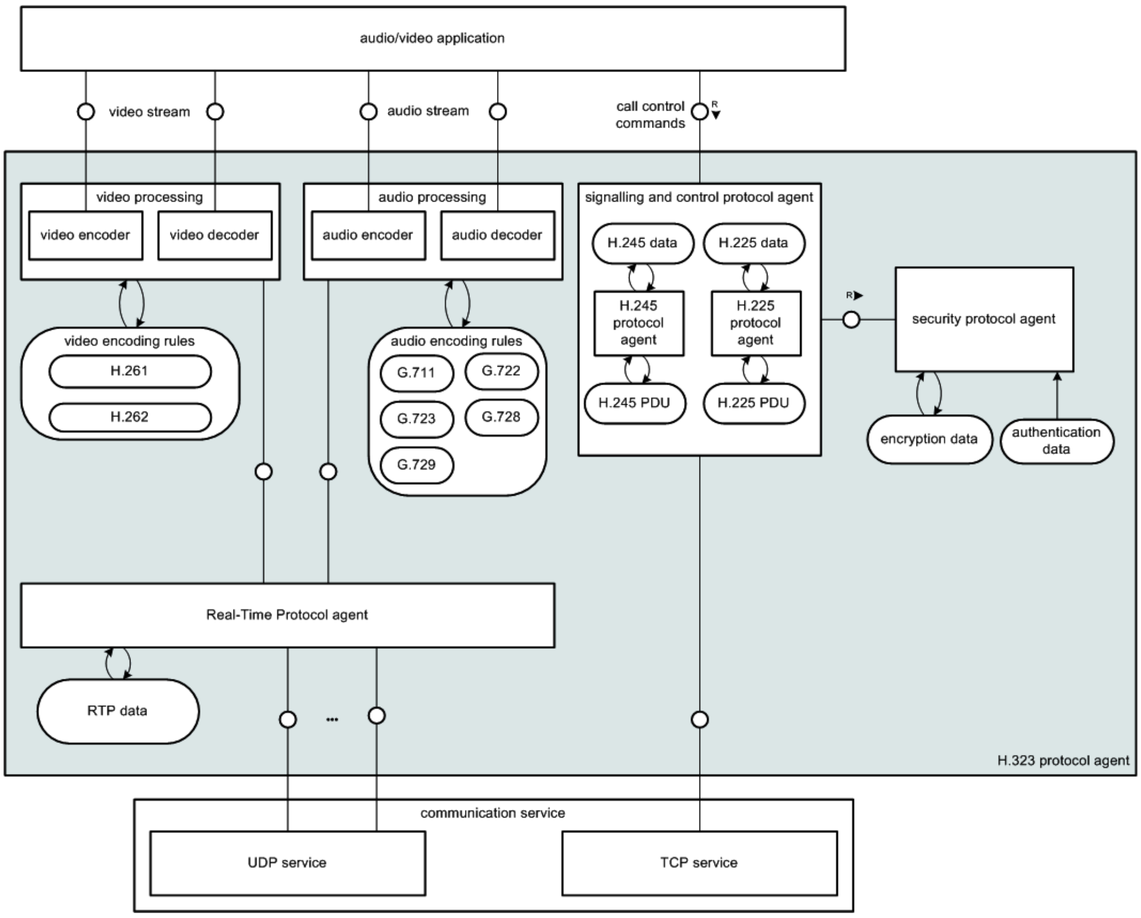


Figure 5.2: H.323 Protocol Architecture with Subprotocols

An overview of the functionality of the protocols presented in Figure 5.2 is given in the following.

Call Signalling and Control Protocols

The H.225.0 protocol does call signalling, creates packets out of the data stream, and implements synchronisation of the data stream. The H.245 protocol describes messages and procedures for establishing and closing of logical channels which are used for transferring audio information, video information and data. In addition, it handles the control of capacities which are needed for the data transfer.

Security Protocols

The H.225.0/RAS protocol allows an endpoint to request authorization to place or accept a call. Further it enables firstly a gatekeeper to control access to and from devices under its control and secondly to communicate the address of other endpoints so that two gatekeepers can easily exchange addressing information. Security mechanism like encryption and authentication are provided by the H.235 protocol.

Protocols for Supplementary Services

H.323 also provides supplementary services like call transfer, call hold, or call diversion. These services are enabled through the H.450 protocol family.

Audio and Video Processing Protocols

For audio processing which means modulation of voice frequencies and coding of speech, the protocols G.711, G.722, G.723.1, G.728, and G.729 are used. In order to process video data the H.323 protocol utilizes the protocols H.261 and H263.

Data Transmission Protocol

The T.120 protocol implements data transmission between end points. It can be used for various applications in the field of collaborative work, such as white-boarding, application sharing, and joint document management.

Transport Protocols

The actual audio and video data is transferred by the Real-time Transport Protocol (RTP) and RTP Control Protocol (RTCP). The latter one carries control information between the communication nodes. Both real time protocols use UDP/IP for transferring their packets. The packets of the call signalling and control protocols are transferred by the TCP/IP protocol.

5.2.2 Usage Scenarios

To present the way the H.323 protocol is working, two usage scenarios are introduced. The first one shows the communiction between two H.323 terminals in the same network. The second scenario describes the communiction between a terminal in the H.323 network and a analogue telefon connected to the PSTN, this scenario also gives an impression of the functionality of the Gatekeeper and the Gateway in the H.323 network. These H.323 call scenarios can be described in five phases:

- Call Setup
- Capability Exchange
- Call Initiation
- Data Exchange

- Call Termination

Basic Usage Scenario

The first usage scenario describes the procedure for placing a call between two endpoints A and B in the same H.323 network. So there is a direct connection where each endpoint is a point of entry and exit of a media flow.

Call Setup

To establish a call between two endpoints requires two channels between the endpoints: one for the call setup and the other one, the so-called signalling channel, for capability exchange and call control. The caller at endpoint A connects to the callee at endpoint B on a well-known port, port 1720, and sends the call *Setup* message as defined in the H.225.0 specification. The Setup message includes:

- message type, in this case, Setup
- bearer capability, which indicates the type of call, for example, audio only
- callee's number and address
- caller's number and address
- protocol Data Unit (PDU), which contains the used version of H.225.0

After endpoint B receives the *Setup* message, it responds with one of the following messages: *Release Complete*, *Alerting*, *Connect*, *Call Proceeding*
In this case, endpoint B responds with the *Alerting* message and endpoint A must receive the Alerting message before its setup timer expires after one minute. Endpoint A opens the signalling channel on a dynamically allocated port at the endpoint B. After sending *Alerting* message, the user at endpoint B must either accept or refuse the call with a predefined time period of less than one second. When the user at endpoint B picks up the call, a *Connect* message is sent to endpoint A and the call setup is completed. The following step is the capability exchange. The protocol directives which are exchanged for setting up the call are shown in Figure 5.3. The call is accepted by endpoint B.

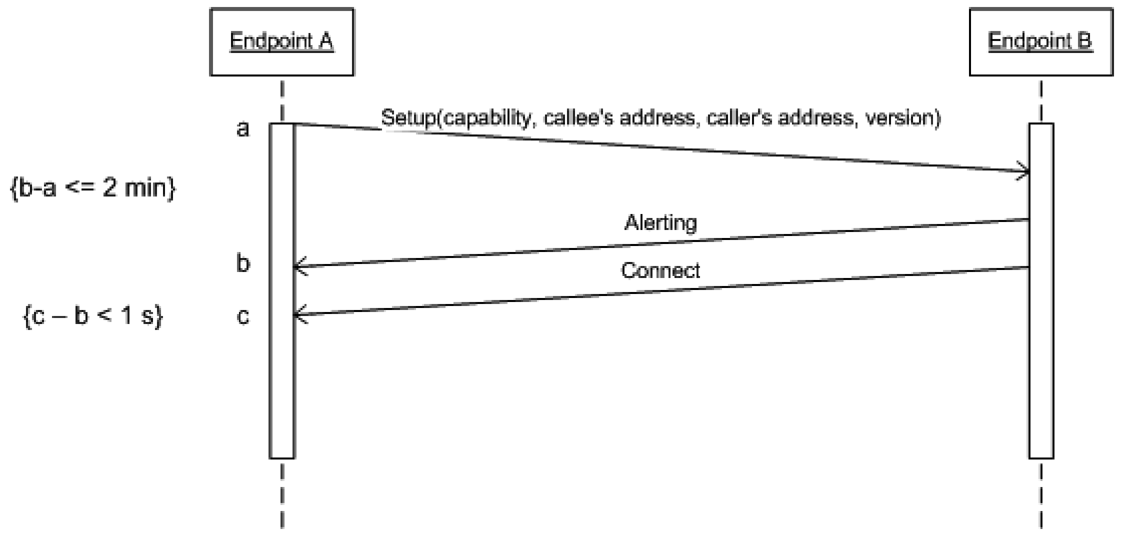


Figure 5.3: H.323 Call Setup

Capability Exchange

The call control and capability exchange messages, as defined in the H.245 standard, are sent on the call control channel. This channel remains active for the entire duration of the call. The call control channel is like the signalling channel unique for each call between endpoints.

An H.245 *TerminalCapabilitySet* (TCS) message that includes information about the codecs supported by that endpoint is sent from one endpoint to the other. Both endpoints send this message and wait for a reply which can be one of the following messages:

- TerminalCapabilitySetAck (TCSAck) - Accept the remote endpoints capability
- TerminalCapabilitySetReject(TCSRej) - Reject the remote endpoints capability

The two endpoints continue to exchange these messages until a capability set that is supported by both endpoints is agreed. In Figure 5.4 the negotiation between both endpoints is illustrated. After succesful negotiation, the next phase *call initiation*, can begin.

Call Initiation

Once the capability setup is agreed, endpoint A and B must set up the voice channels over which the voice data (media stream) will be exchanged. A sends an H.245 *Open-LogicalChannel* message to endpoint B, this message specifies the type of data being

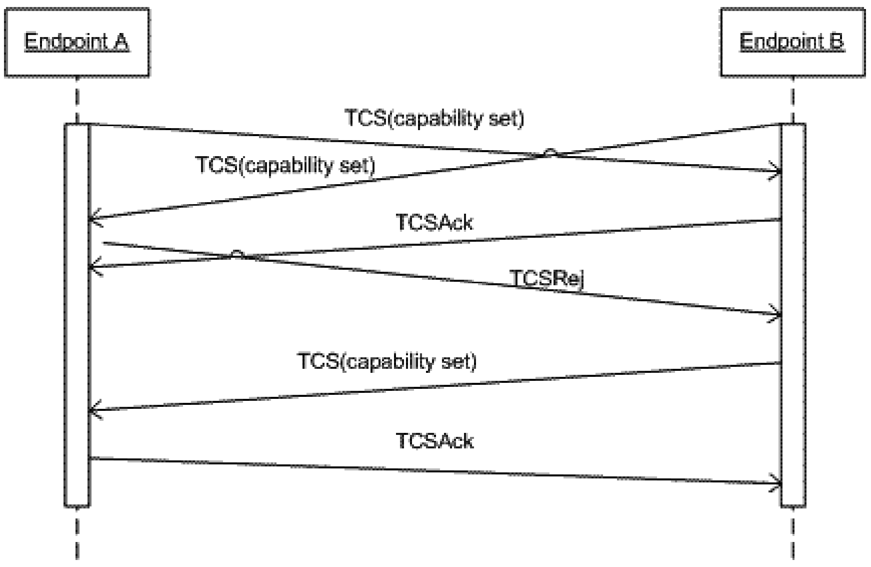


Figure 5.4: H.323 Capability Exchange between two Endpoints

sent, for example, the codec that will be used. For voice and/or video data, the message also includes the port number that endpoint B should use to send RTCP receiver reports. After endpoint B is ready to receive data, it sends an *OpenLogicalChannelAck* message to endpoint A. This message contains the port number on which endpoint A has to send RTP data and the port number on which endpoint A should send RTCP data. Endpoint B repeats the process above to indicate which port endpoint A will receive RTP data and also send RTCP reports. This initiation process is shown in Figure 5.5. After these ports have been identified, the next phase *data exchange* starts.

Data Exchange

Endpoint A and endpoint B exchange information in RTP packets that carry the voice and/or video data. During this exchange both sides send periodically RTCP packets, which are used to monitor the quality of the data exchange. When the data exchange has been completed the phase *call termination follows.*

Call Termination

To terminate an H.323 call, one of the endpoints e.g. endpoint B hangs up. Endpoint B must send an H.245 *CloseLogicalChannel* message for each logical channel it has opened with endpoint A. Accordingly, endpoint A must reply to each of those messages with a *CloseLogicalChannelAck* message. When all the logical channels are closed, endpoint B sends an H.245 *EndSessionCommand* and waits until it receives the

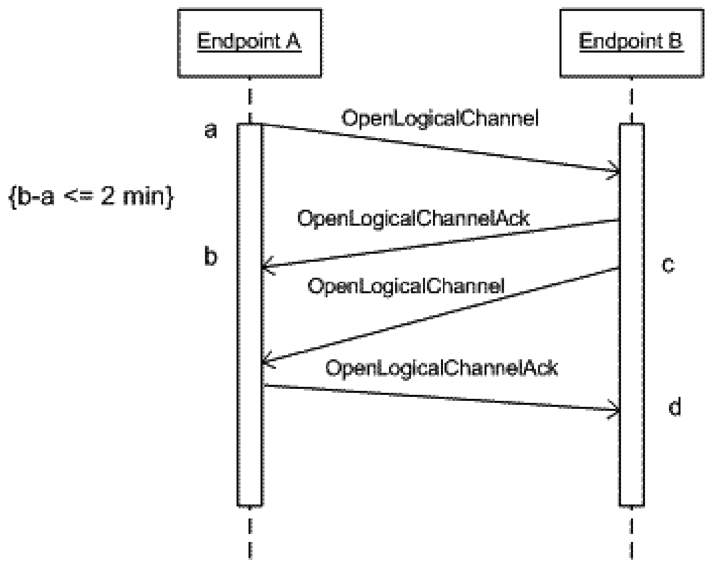


Figure 5.5: H.323 Call Initiation

same message from endpoint A and after that closes the call control channel. Both endpoint A and endpoint B then send an H.225.0 *ReleaseComplete* message over the signalling channel, which closes that channel and ends the call. Figure 5.6 shows the described termination of a call.

Usage Scenario including Gateway

This scenario shows the usage of the gateway which provides a bridge with other networks, for example, an IP network or the PSTN. In this scenario user A is at a terminal in the H.323 network, while user B is by a phone connected to the PSTN. The gatekeeper, as described in Section 5.1.3, provides network services such as Registration, Admission and Status (RAS) and address mapping. When a gatekeeper is present, all endpoints managed by the gatekeeper must register with the gatekeeper at start-up.

Call Setup

The user at endpoint A attempts to locate a gatekeeper by sending out a *Gatekeeper Request* (GRQ) message and waiting 5 seconds for a response. When endpoint A receives a *Gatekeeper Confirm* (GCF) message, the endpoint registers with the Gatekeeper by sending the *Registration Request* (RRQ) message and waiting 3 seconds for a *Registration Confirm* (RCF) message. If more than one gatekeeper responds, endpoint A chooses only one of the responding gatekeepers. After the registration the endpoint requests permission to call from the gatekeeper. Therefore, it sends an *Admission*

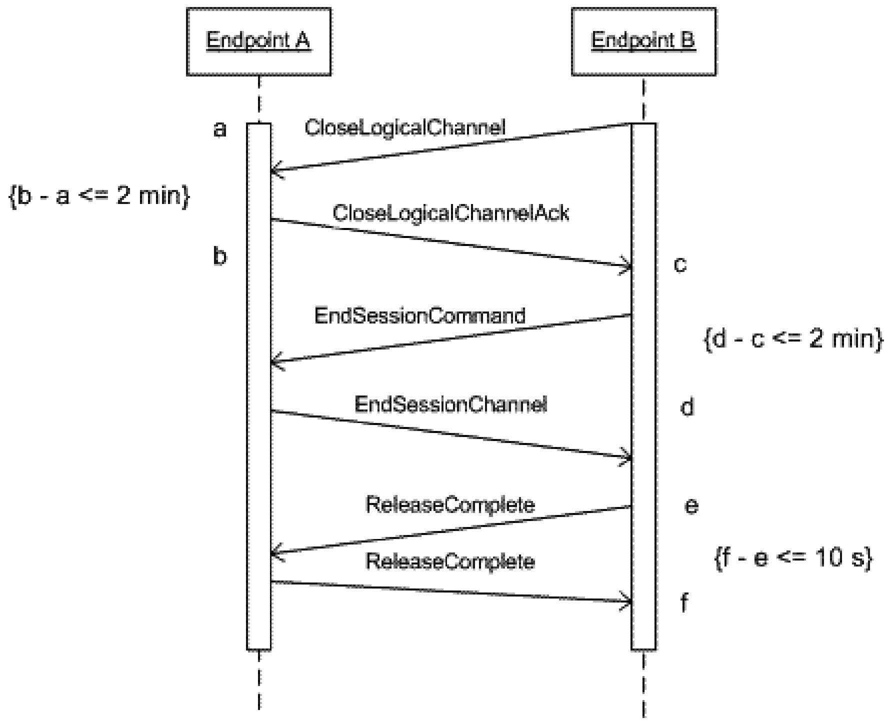


Figure 5.6: H.323 Call Termination

Request (ARQ) message to the gatekeeper. This message includes information such as:

- sequence number
- gatekeeper assigned identifier
- type of call, in this example, point-to-point
- call model to use, either direct or gatekeeper-routed
- destination address, in this case, the phone number of endpoint B
- estimation of the amount of bandwidth required. This parameter can be adjusted later by a Bandwidth Request (BRQ) message to the gatekeeper.

If the gatekeeper allows the call to proceed, it sends an Admission Confirm (ACF) message to endpoint A. This ACF message includes the following information:

- call model used
- transport address and port to use for call signalling
- allowed bandwidth

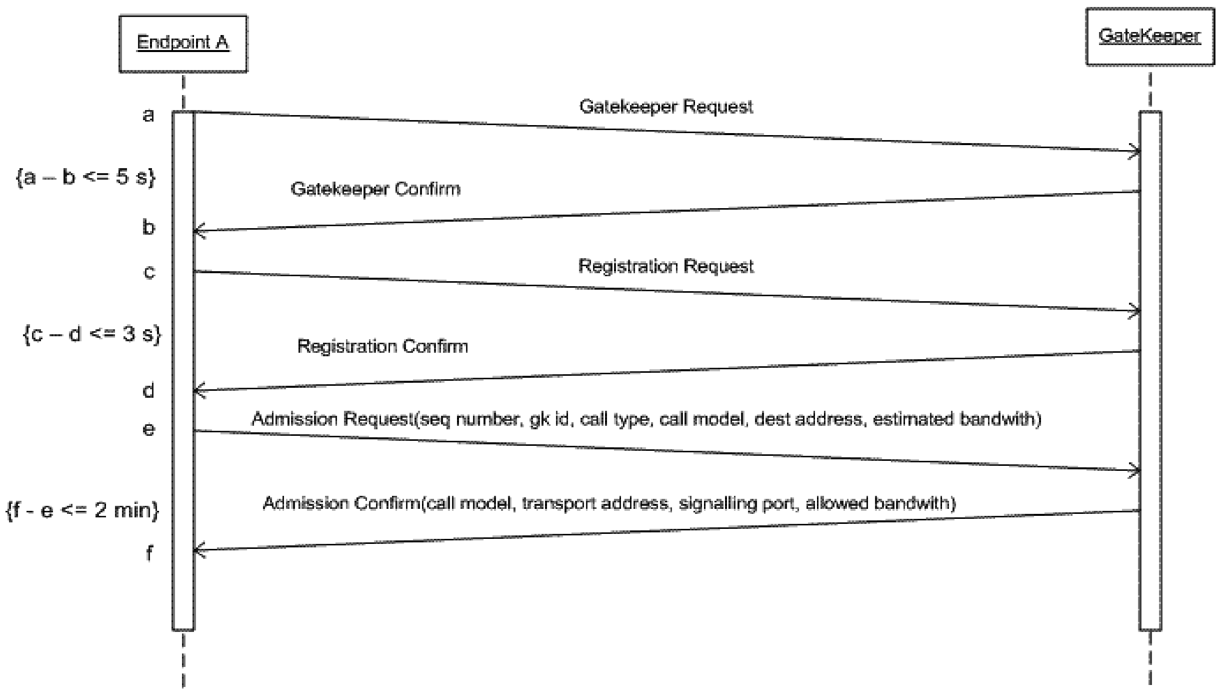

Endpoint A
GateKeeper
a
{a – b <= 5 s}
b
c
{c – d <= 3 s}
d
e
{f - e <= 2 min}
f
Gatekeeper Request
Gatekeeper Confirm
Registration Request
Registration Confirm
Admission Request(seq number, gk id, call type, call model, dest address, estimated bandwith)
Admission Confirm(call model, transport address, signalling port, allowed bandwith)

Figure 5.7: Message exchange for Registration and Admission

The exchange of registration and admission messages is shown in Figure 5.7. Now the endpoint A can initialize the connection to the gateway. Therefore, it sends the *Setup* message to the gateway. Since the destination phone is connected to an analogue line (the PSTN), the gateway goes off-hook and dials the phone number using dual tone multifrequency (DTMF) digits. The gatekeeper, therefore, is converting the H.225.0 signalling into the signalling present on the PSTN. Depending on the location of the gateway, the number dialled may need to be converted. For example, if the gateway is also located in Europe, then the international dial prefix will be removed. As soon as the gateway is notified by the PSTN that the phone at endpoint B is ringing, it sends the H.225.0 *Alerting* message as a response to endpoint A. When the phone is picked up at endpoint B, the H.225.0 *Connect* message is sent to endpoint A by the gateway. As part of the *Connect* message, a transport address that allows endpoint A to negotiate codecs and media streams with the gateway as B's proxy is sent. The described establishing of the connection is shown in Figure 5.8

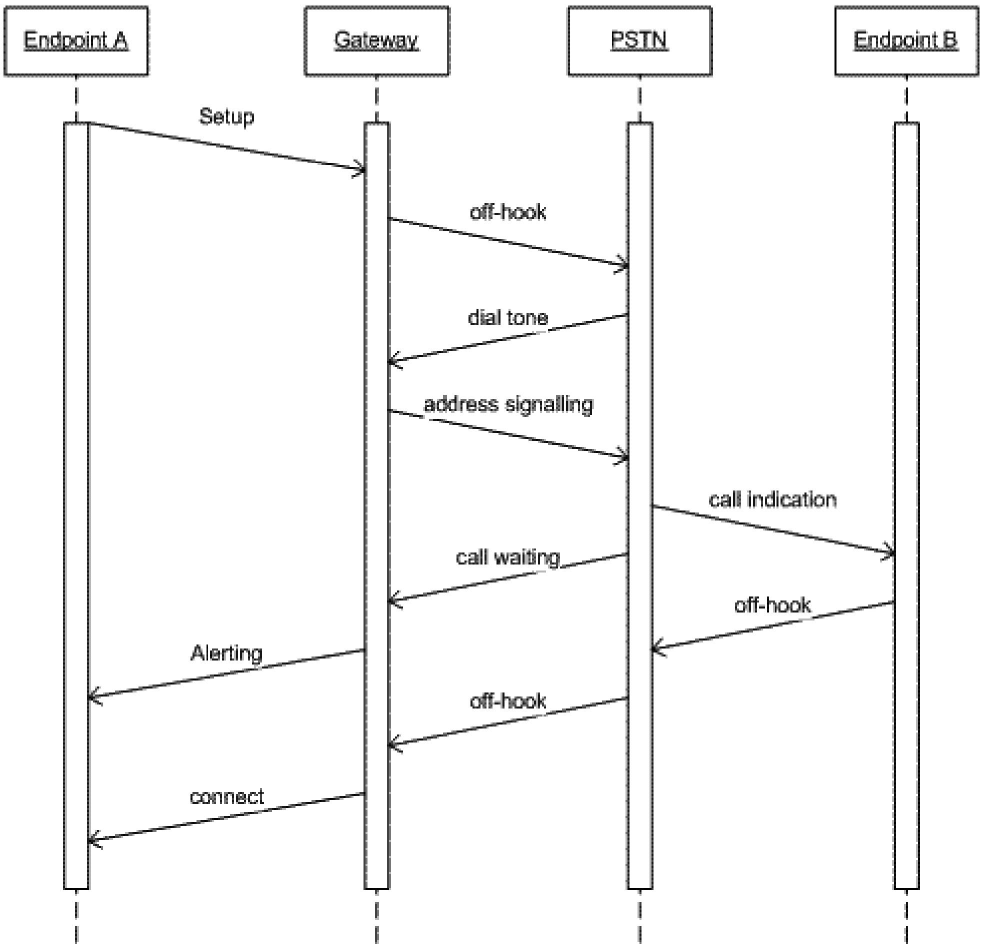


Figure 5.8: Establishing Connection between H.323 Endpoint and Telephon in PSTN

The H.225.0 and H.245 signalling used in the phases *capability exchange*, *call initiation* and *exchange data* are the same as that described in the basic H.323 call scenario above. Endpoint A is communicating via H.323 only with the gateway which is converting the H.323 protocol directives to directives used by the PSTN. Call Termination As in the basic H.323 call scenario the endpoint which terminates the call first, needs to close all the channels that were open using the H.245 *CloseLogicalChannel* message. If the gateway terminates first (actually endpoint B), it sends an H.245 *EndSession-Command* message to endpoint A and waits for the same message from endpoint A. The gateway then closes the call control channel. Afterwards by exchangign *ReleaseC-omplete* messages the signalling channel is closed. The closing of all communiction channel is shown in Figure 5.9.

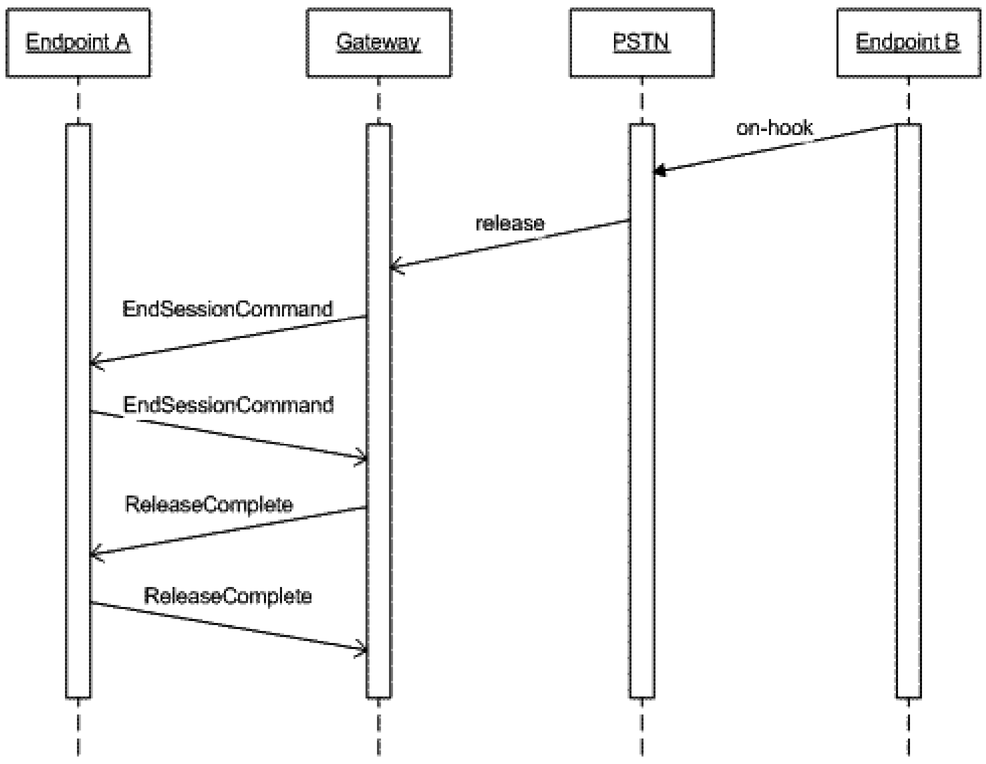


Figure 5.9: Closing of Signalling Channel between endpoint and Gateway

When all channels between endpoint A and the gateway are closed, each must send a *DisengageRequest* (DRQ) message to the gatekeeper. This message lets the gatekeeper know that the bandwidth is being released. The gatekeeper sends a *DisengageConfirm* (DCF) message to both, endpoint A and the gateway, as illustrated in Figure 5.10.

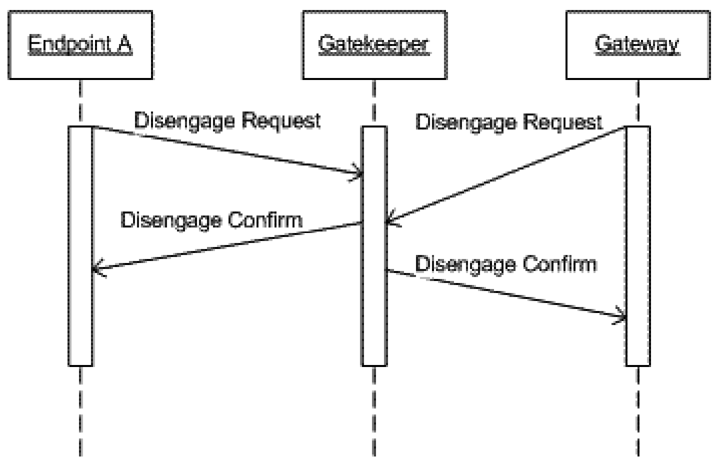


Figure 5.10: Messages Exchange for Releasing Bandwidth

5.3 OpenH.323 protocol Software Architecture

5.3.1 Conceptual View of System Structure and Behaviour

In the following, the OpenH.323 protocol software architecture is described by a the conceptual view which means from high-level structure. The conceptual view is described in Section 2.2.

The OpenH.323 architecture is a typical peer-to-peer architecture which means every participant in the network can provide services to others and use certain services which are provided by other participants.

In the OpenH.323 architecture, a participant is represented by the endpoint manager as shown in Figure 5.11. This component is able to initiate and receive several connections concurrently, which requires multithreading. Figure 5.11 shows only one created connection because of readability but in general the endpoint manager can create as much as necessary. The endpoint manager supports the different H.323 component roles which are described in section 5.1. Which role the certain endpoint manager supports is defined in the type information.

Incoming connection requests are received by the TCP/IP communication service of the operating system and handed over to the endpoint's listener. The listener accepts the connection so the communication service can establish the TCP/IP connection and create a new socket data structure which is used for the connection.

After accepting the connection request, the listener is waiting for further incoming requests. The listener's behaviour is shown in Figure 5.12.

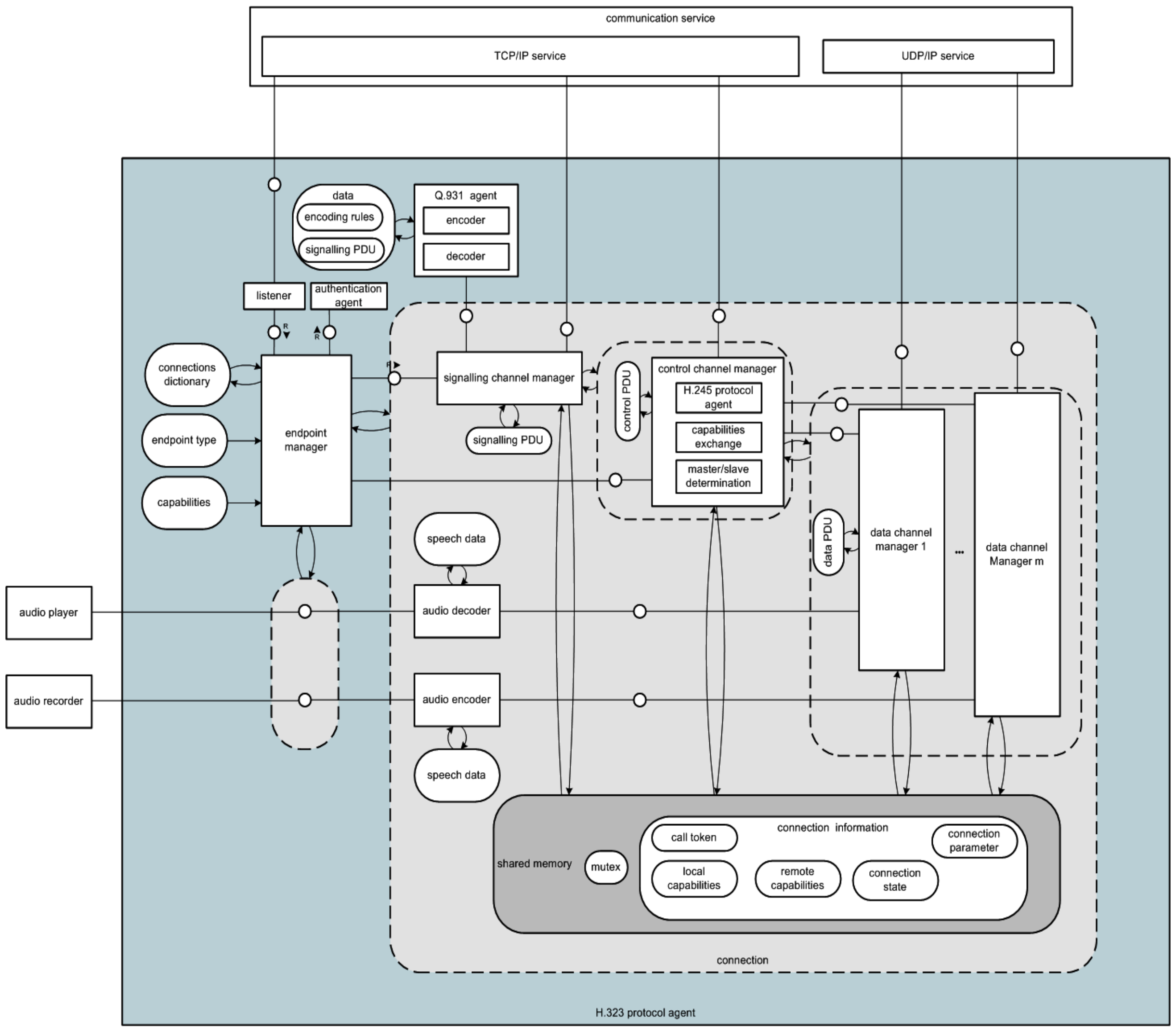


Figure 5.11: Conceptual view on the OpenH.323 protocol software architecture

As shown in Figure 5.11 the endpoint manager creates a new connection with a shared memory for the connection's information and also the signalling channel manager which is a thread. Then the endpoint manager adds the connection to its connection dictionary. This described behaviour is also illustrated in Figure 5.12.

The signalling channel manager is using socket, returned by the TCP/IP communication service, to communicate via H.225 protocol, which is described in Section 5.2.1. The main task of the signalling manager is the exchange of call signalling information with the communication partner's signalling channel manager. When requested, the signalling channel manager creates a new thread, the control channel manager which is responsible for call control and creation of data channel managers which are also shown in Figure 5.11. The control channel manager has two possibilities to communicate with the communication partner's control channel manager. Either it can use an own TCP/IP socket or it uses the same socket as the signalling channel manager but then a tunneling mechanism must be available. The default setting is the own TCP/IP socket.

The call's data is transmitted and recieved by the data channel managers. These components are also single threads which are created by the control channel manager, which is also responsible for terminating the data channel managers. Each data channel manager uses a UDP/IP socket for recieving or transmitting data. The data channel manager is able to support different real-time transport protocols, for example RTP which is used by default. In the RTP case, the data channel manager is responsible for adding or deleting the RTP header. The data channel manager can either be a reciever or transmitter. A reciever writes the data which it recieves to the decoder which sends the decoded data to the playing device. A transmitter sends data which it reads from the encoder. The encoder encodes the data it recievied from the recording device. The operations which are performed by the data channel managers are shown in figure 5.13. The channels between devices and decoder/encoder are established by the endpoint manager when he gets a request from the control manager.

The petri nets in Figures 5.12 and 5.13 show the interaction between the conceptual view's components while enabling and performing a call.

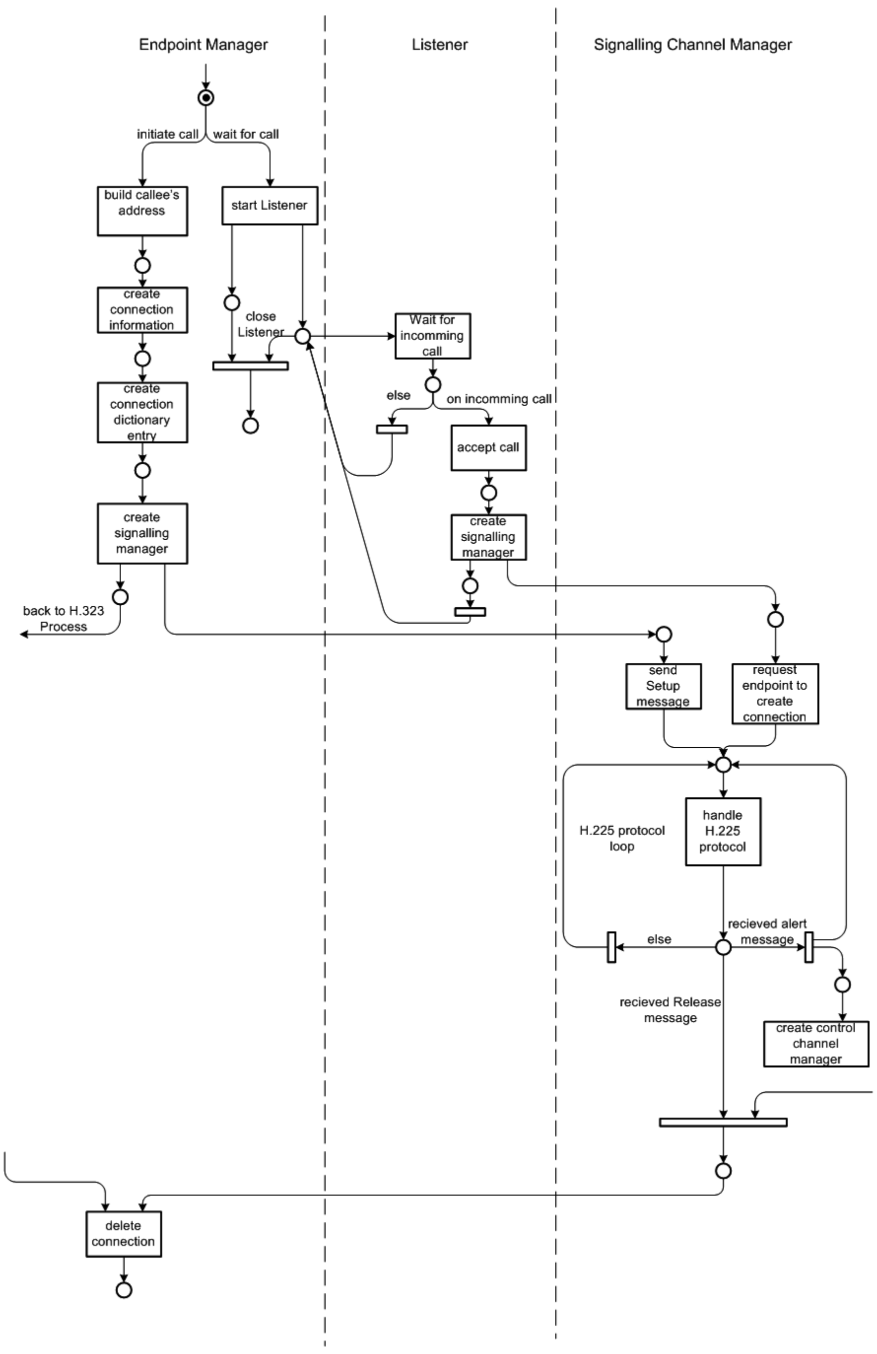


Figure 5.12: Interaction of main components during call

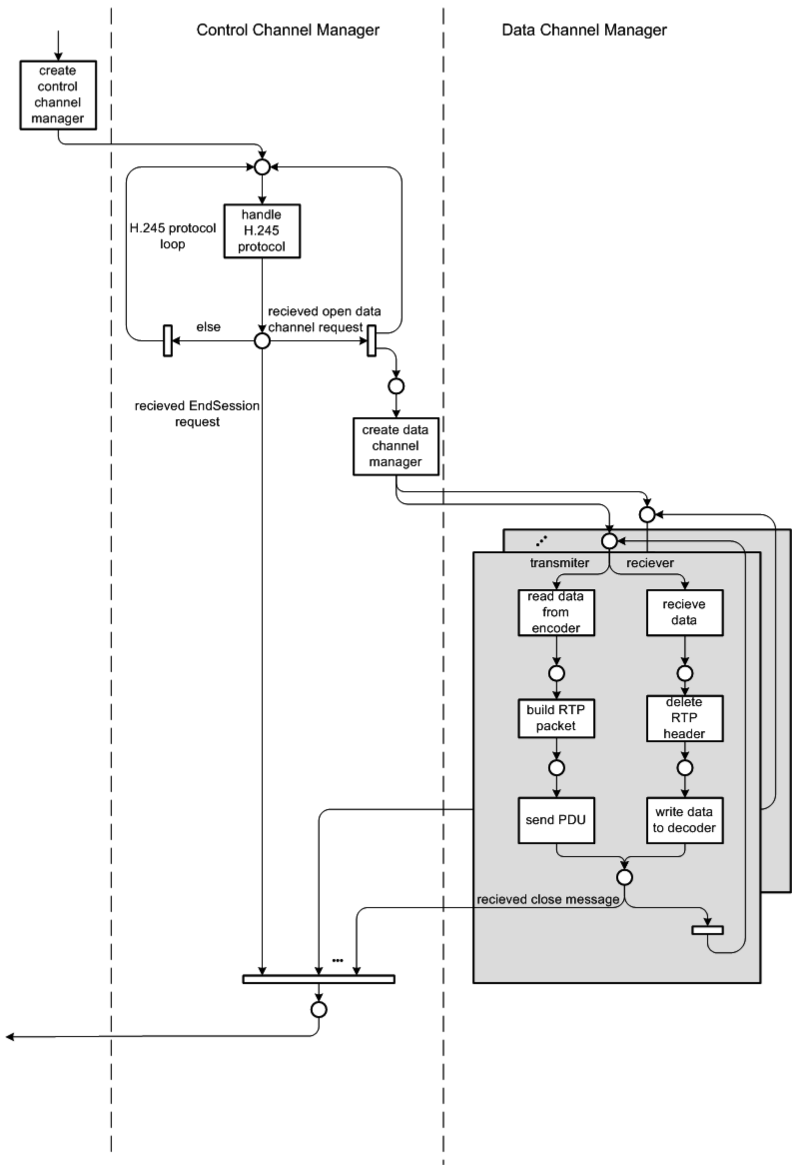


Figure 5.13: Interaction of main components during call (cont.)

The signalling channel manager and the control channel manager implement the main functionality of the OpenH.323 protocol. Therefore, the operations which they perform in their loops are described in more details. These loops are parts of the petri nets which are shown in Figures 5.12 and 5.13. Figure 5.15 shows the way the signalling channel manager handles and processes H.225 requests.

The operations which are performed by the control channel manager in its loop are described in Figure 5.14. This component negotiates the call's capabilities, determines whether caller or callee act as master or slave, controls the call's parameter, e.g. jitter and bandwith, and manages the creation and termination of data channel managers. Jitter is the variation in the delay of the packets arriving at the receiving end. Reasons for Jitter are for example congestion at various points in the network, varying packet sizes that result in irregular processing times of packets, and out of order packet delivery.

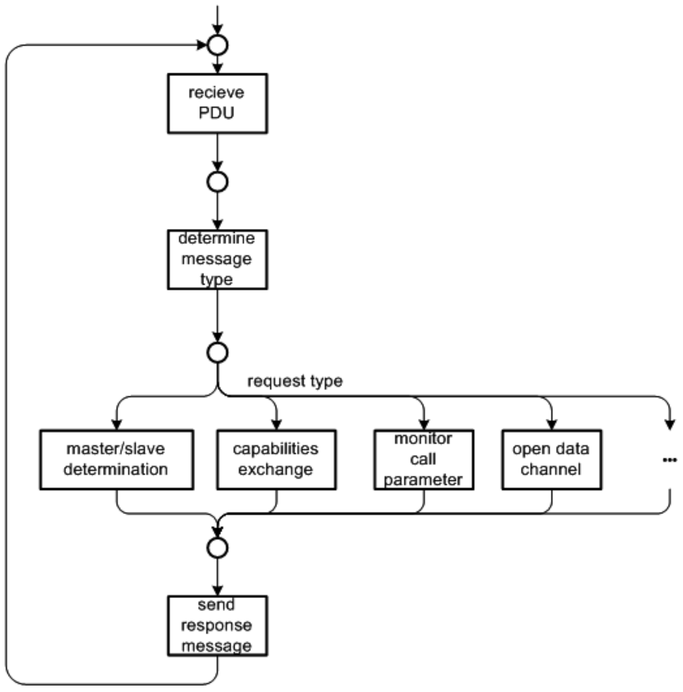


Figure 5.14: H.245 protocol loop

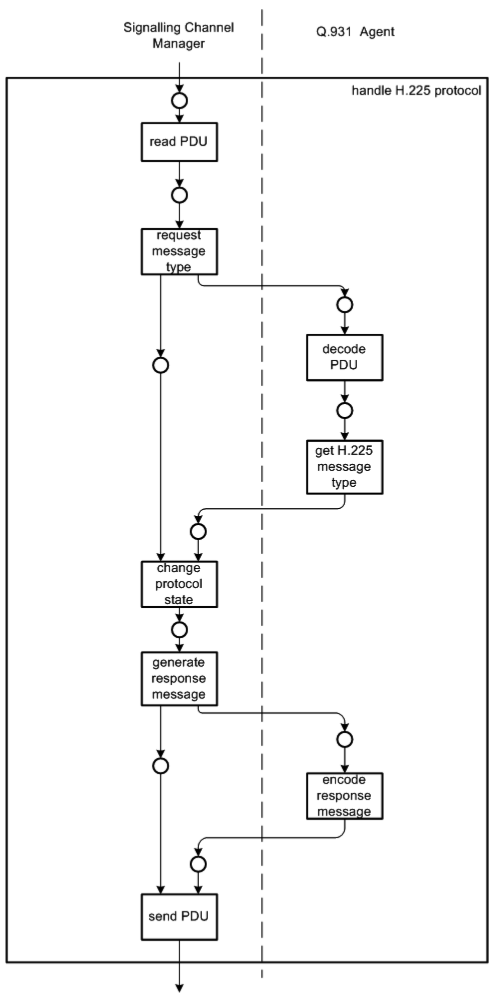


Figure 5.15: Handling of H.225 protocol

5.3.2 Logical and Process View

In this section the OpenH.323 protocol software architecture is described with Krutchen's logical and process view [9] as described in Section 2.2. So this description is quite close to the code and considers implementation details.

Figure 5.16 shows the OpenH.323 protocol software architecture as UML class diagram. The system is decomposed into classes which implement its main functionality. In the

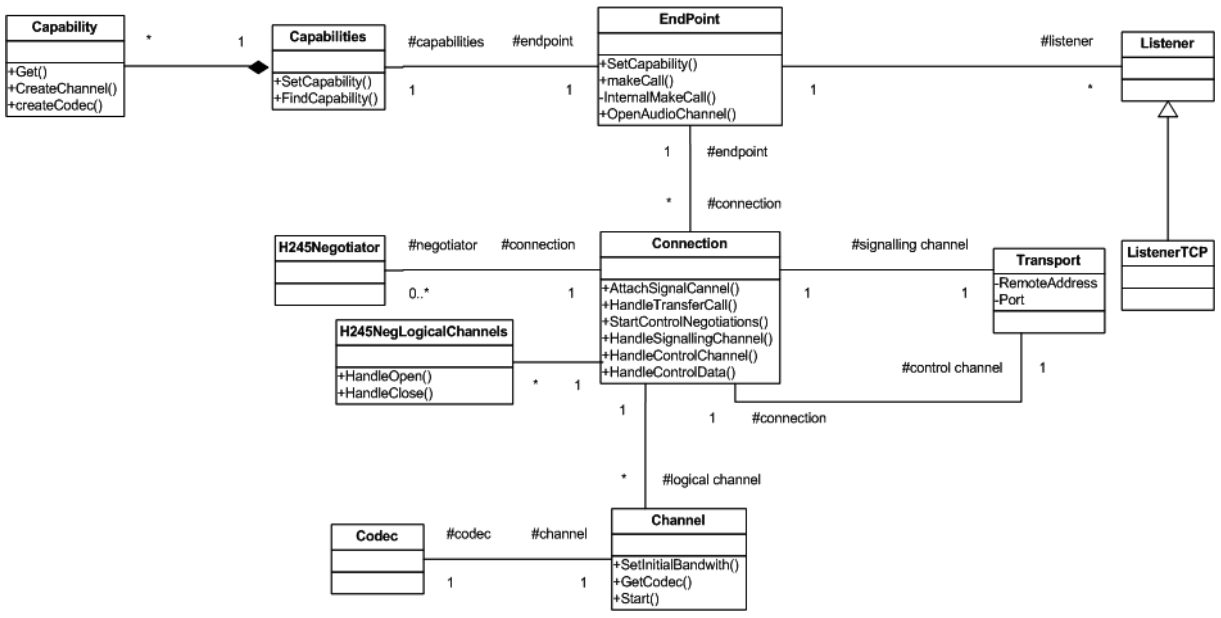


Figure 5.16: Static class structure

following the classes shown in Figure 5.16 was described regarding their functionality and their relation to the conception view which is describe in the previous Section 5.3.1.

An application which is utilizing the OpenH.323 protocol would typically have one instance of a descendant of the *H323Endpoint* class. This class implements the functionality of the endpoint manager component which is described in the conceptual view in Section 5.3.1. The descendant class would set up defaults for various H323 parameters, the most important of which is the capability table which defines the codecs and channel types the application is capable of handling. Also created by the application in the *H323Endpoint* would be instances of one or more descendants of the *H323Listener* class. There are descendant classes for the TCP, UDP, and IP protocol. A listener spawns a thread that monitors its protocol and when a new incoming call is detected, creates an instance of a *H323Transport* class descendent. The *H323Transport* class contains attributes for address information which are IP address and port number. As for the *H323Listener* class, there is a descendent for each protocol supported, e.g. *H323TransportTCP*.

In case the application recieves an incoming call or initiates an outgoing call the Endpoint class creates a new connection and H.323 signalling and negotiation begin. The connection is embodied by the *H323Connection* class, which contains all of the state information for a connection between H.323 endpoints. As well as for signalling and negotiations a thread is spawned. The operations performed by these threads correspond with the behaviour of the conceptual view's signalling channel manager and the control channel manager which are both describe in the Section 5.3.1. The operations for the signalling are embodied by the *H323Connection* class. The H.323 negotiations are part of the H.245 protocol. The *H323Negotiator* classes are used to maintain the state and functionality of each command or variable defined by the H.245 protocol. Figure 5.16 shows the *H323Negotiator* classes.

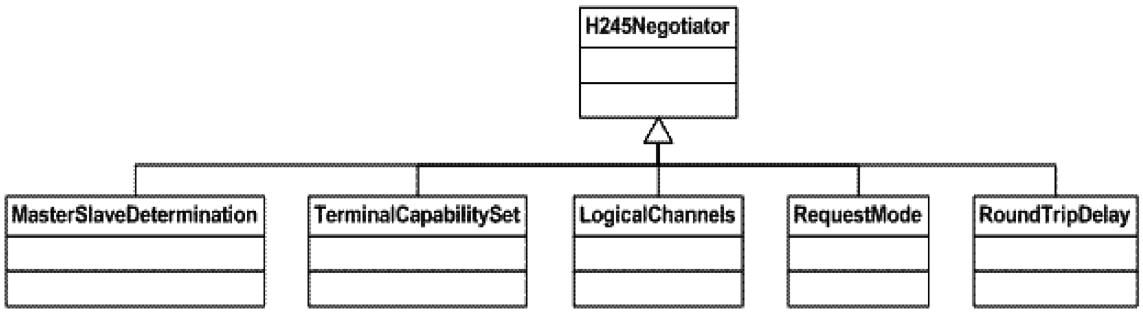


Figure 5.17: Inheritance hierarchy of H323Negotiator class

The H.245 negotiations may result in the creation of logical channels as well as by the remote endpoint and by the local application. The *H323Channel* class descendants represents a logical channel. A typical use of one of these classes is to open a stream of encoded audio data. The *H323Channel* class would create a *H323Codec* using the *H323Capability* that was passed during the protocol negotiations. Each instance of a logical channel has a single thread performing the channels operations, the threads correspond with the data channel managers from the conceptual view 5.3.1. Starting and terminating these threads is the responsibility of an instance of the *H245NegLogicalChannels* class.

The main purpose of the OpenH.323 protocol is to facilitate calls. In the Appendix A, UML sequence diagrams are given which show the interaction of objects, which are instantiated from the classes shown in Figure 5.16, to realize a call between two H.323 endpoints. These diagrams show sequences of executed operations at runtime so they are meant to represent the OpenH.323 protocol software architecture from Kruchten's process view [9]. All operations are executed in the application's process that means they are elementary units of control. The following Table 5.1 shows the mapping between the conceptual and logical view.

Conceptual Component	Class in the Logical View
endpoint manager	Endpoint class Capabilities class Capability class
signalling manager	Connection class Transport class
control manager	Connection class H245Negotiator class H245NegLogicalChannels Transport class
data channel manager	Channel class Transport class
connection information	Connection class
audio encoder	Codec class
audio decoder	Codec class
listener	Listener class

Table 5.1: Mapping of Conceptual and Logical View

6 Evaluation of the OpenH.323 Protocol Architecture

The evaluation of the OpenH.323 protocol architecture will focus on evaluating the three quality characteristics efficiency, maintainability, and security. These three have been identified as important characteristics for the system's stakeholders as described in Section 3.2 and shown in the role-base quality model in Figure 3.2.

The refinement processes of these quality characteristics which are shown in Figures 3.4 and 3.3 illustrates that the efficiency evaluation is based on external metrics (Section 3.1.1) which are collected while testing the running system. The maintainability can be evaluated with the help of internal metrics (Section 3.1.1) which are measured in the source code. The quality characteristic security can be evaluated by analyzing the utilized security mechanisms.

Since the evaluation inputs are the architectural description which is shown in Section 5.3 and the source code, the suitable evaluation methods, which can be utilized, are: the early evaluation methods which are described in Section 4.1.1 and the collection of architectural metrics which are described in section 4.3.4. But the focus is on the early evaluation because the only existing artefacts are the conceptual and the logical description of the software system. These descriptions cannot be evaluated quantitatively regarding quality characteristics. Among the early evaluation methods, the ATAM method is chosen, which is described in Section 4.3.3. It reveals how well an architecture satisfies particular quality goals and recognizes that architectural decisions affect more than one quality attribute, so this method enables the identification of trade-offs among several quality attributes. Furthermore, the architectural metrics approach which belongs to the late evaluation methods will be used to retrieve quantitative values to evaluate the maintainability. Only the collection of architectural metrics is used because the source code of the system is available. That means the evaluation results of the early evaluation regarding maintainability can be substantiated. The architectural metrics do not consider any relations to the characteristics efficiency and security. They only give the possibility to evaluate maintainability.

6.1 Early Evalution with ATAM

The ATAM evaluation as described in Section 4.3.3 will be adjusted to my special case because the evaluation is performed by one person acting in the roles described in Table 4.1. This paper already described the architecture in Section 5.3 and the three quality characteristics which are of interest have been identified. So the next step is the identification of the most important scenarios and the description of the utility tree.

The purpose of OpenH.323 protocol software is calling, so the most important scenario is the call scenario. This paper focuses on the evaluation of the basic call scenario which is described in Section 5.2.2. This scenario can be devided into smaller scenarios according to the quality requirements.

The efficiency requirements in the basic call scenario are especially the time limits for exchanging the protocol directives. These time limits are shown in the sequence diagrams which are used for describing the basic call scenario in Section 5.2.2. Further the architecture's ability to process and transfer audio/video data with a low end-to-end delay (less than 200 millisec ([32])) is important fact which has to investigated.

Regarding maintainability especially the ability to perform changes to the architecture is interesting because changes or extensions to the existing protocols and codecs are thinkable. That means the identification of architectural elements affected by a change and the identification of the changes in the relationships between architecture components is necessary.

Security is evalutated by investigating the used mechanisms for authorization, authentication, and encryption.

6.1.1 Utility Tree

The utility tree for the OpenH.323 protocol architecture is shown in Figure 6.1. Usually, the scenarios are prioritized as described in Section 4.3.3, but the scenarios which are given in this utility tree are the most important one to evaluate the quality characteristics efficiency, maintainability, and security for the protocol architecture.

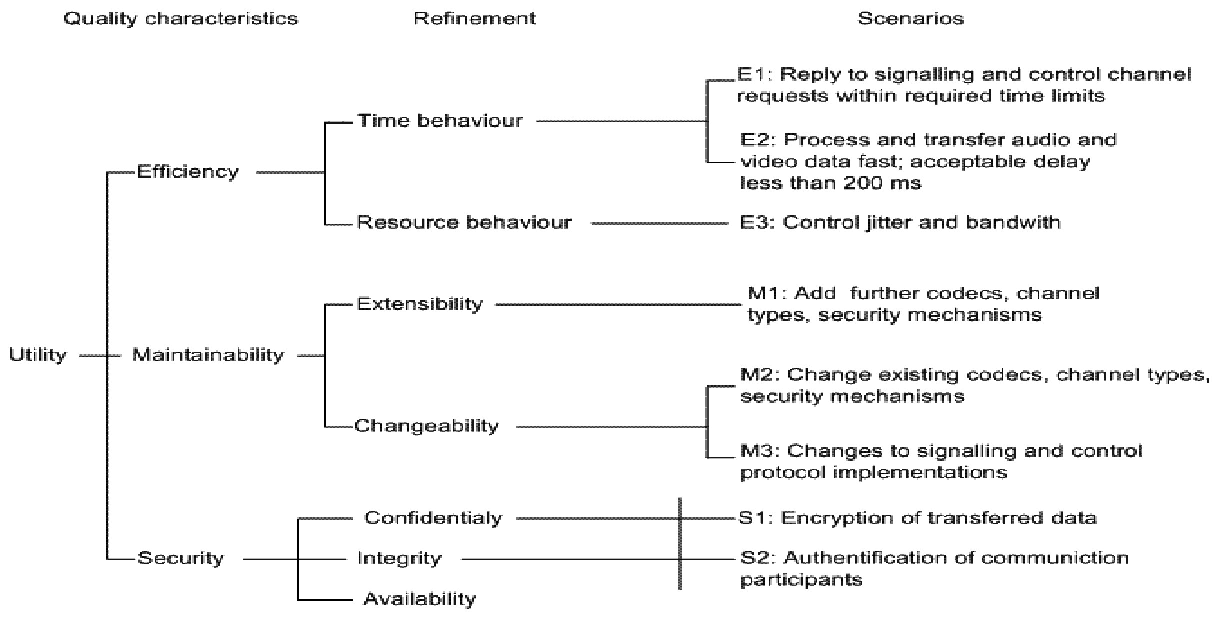

Quality characteristics
Refinement
Scenarios
Utility
Efficiency
Maintainability
Security
Time behaviour
Resource behaviour
Extensibility
Changeability
Confidentialy
Integrity
Availability
E1: Reply to signalling and control channel requests within required time limits
E2: Process and transfer audio and video data fast; acceptable delay less than 200 ms
E3: Control jitter and bandwith
M1: Add further codecs, channel types, security mechanisms
M2: Change existing codecs, channel types, security mechanisms
M3: Changes to signalling and control protocol implementations
S1: Encryption of transferred data
S2: Authentification of communiction participants

Figure 6.1: Utility tree for OpenH.323 protocol evaluation

6.1.2 Realisation of Scenarios by the Architecture

In the following, the scenarios which are shown in the utility tree in Figure 6.1 will be investigated regarding the architectural design decisions, trade-offs, and risks.

Efficiency Scenarios

Scenario E1

The first efficiency scenario E1 requires that incoming requests over the signalling and control channel are processed and responded within the required time limits. The time limits are given in the sequence diagrams in Figures A.3 through A.2. To process incoming requests the OpenH.323 architecture has two concurrent working components: the signalling channel manager and the control channel manager, which both are shown in Figure 5.11. These components are realised by threads. Multithreading creates a certain overhead which is caused by the context switches from one thread to another. The data channel manager are also implemented by threads which means within the OpenH.323 process there are many context switches. So the efficiency suffers from that overhead. The operations which are necessary to process requests are shown in the signalling loop in Figure 5.15 and in the control loop in Figure 5.14. Since the operations of these loops are mapped to the operations of the logical view's connection class (Figure 5.16) the threads, which implement the signalling channel manager and the control channel manager, have write access to the connection information (Figure 5.11) which means the connection data is located in a shared memory. Access to this shared memory must be synchronised to avoid inconsistent data. This synchronisation is achieved with a mutex mechanism which is also shown in Figure 5.11. But that also means it is a bottleneck because only one thread can access the connection data at a certain time. That means all other threads have to wait until they are allowed to access. Hence, this mutex mechanism is negatively affecting on the system's efficiency.

To evaluate whether the architecture can meet the protocol's time limit requirements the sequence diagrams in Figures A.3 - A.2, which illustrate the message exchange between the objects at runtime, can be analysed. What becomes obvious on the first glance is that the objects exchange many messages (ca. 120) for initiating a call. That definitely affects the efficiency of the system negatively. But the time requirements are not difficult to meet because the signalling manager and control manager wait for a maximum period of two minutes (the ITU suggests three minutes) for responses of the communication partner. So even if there is a quite large communication overhead within the architecture and the dependency on the underlying network within two minutes the response should be arrived at the communiction partner. For example, to instantiate a call one signalling manager sends a Setup message, which is shown in Figure 5.3, to the other signalling manager. Figure A.4 shows that only four operations of the connection object are necessary to process the incoming Setup message and to send the reply. These four operations are *HandleSignallingChannel()*, *HandleSignalPDU()*, *OnReceivedSignalSetup()*, and *AnsweringCall()*.

Only in case of registration and admission at the gatekeeper, the gatekeeper's responses must arrive within three and five seconds. The analysis of the request and the creation of the response is done by an gatekeeper agent whose functionality is implemented though a single class *gkserver*. So there is not much communiction with other components at runtime for processing and responding to registration and admission requests. Table 6.1 summarizes the main desing decisions, which affect the first efficiency scenario E1, in relation to trade-offs and risks they cause.

Design Decisions	Trade-offs	Risks
multithreading	efficiency(-)	context switches cause overhead which decreases the system's efficiency
shared memory	efficiency(-) maintainability(-)	- synchronisation of access leads to bottleneck
high degree of decomposition causes high communication degree between objects at runtime	efficiency(-) maintainability(+)	overhead caused by communication between objects

Table 6.1: ATAM output for the scenario E1

Scenario E2

The secenario E2 demands fast processing and transfer of audio and video data to assure a good quality of speech and video. According to [32], a maximum end-to-end delay of 200 milliseconds can be accepted, because for higher delays the quality of the transmitted media streams is very poor. The speech and video data is captured by an recording device (Figure 5.11) and written to an encoder (Figure 5.11 which encodes the data following a certain encoding rule given through a codec. The encoded data is written to a data channel manager (Figure 5.11) which transfers it to the communication partner via RTP protocol. Every data stream is transferred by a single data channel manager which is implemented through a thread. Incoming speech or video data is recieved by a data channel manager which writes it to the decoder. The decoded data is written to a playing device. So the cruatial tasks which affect the quality of the media streams are encoding/decoding and the data transfer.

The OpenH.323 architecture can use different speech codecs. The used codec depends on the available bandwidth for transferring the data. The existing audio codecs are shown in Figure 5.2.1. These codecs are used to digitalize the analogue speech signal. The codec which assures the highest speech quality is G.711 which uses Pulse-Code-Modulation (PCM) for encoding. This codec is also used for ISDN telefony and needs a bandwith of 64 kbit/s. If the data is transferred over IP, like it is the case for

OpenH.323, 80 to 100 kbit/s are necessary to assure a good speech quality because of the overhead created by IP headers. All other available G.XX codecs need lower bandwith, the G.729 codec requires the lowest bandwith because it operates with 8 kbit/s. But of course, the speech quality is also lower for codecs with lower bandwith demandings.

For compression of the video data the architecture can use to either the H.261 or the H.263 codec. Since both codecs are intended for being used for video conferencing they both are optimized for low data rates and relatively low motion. The main difference between both is the video quality which is higher for the H.263 codec. The codecs compress the video data in real-time.

Next to the data encoding also the data transfer is crucial for assuring a good quality of the audio and video stream. To achieve an end-to-end delay of less than 200 milliseconds ([32]), the data channel managers (Figure 5.11), which are responsible for transmitting the streams, utilize the Real-Time Protocol (RTP). RTP is built on top of the User Datagram Protocol (UDP). Since Applications using RTP are less sensitive to packet loss, but typically very sensitive to delays, UDP is more suitable than the Transmission Control Protocol (TCP) for such applications.
The advantage of UDP is that this protocol is connectionless, so it needs less header information and does not use mechanism like retransmission or congestion control which can lead to transmission delays.

Actually, RTP does not provide any mechanism to ensure timely delivery or provide other quality-of-service guarantees. The three most important informations included in the RTP header are:

- sequence number
- timestamp
- used data format (e.g. PCM)

The sequence number is useful for reconstructing the incoming packets in the right order. The timestamp is needed to determine transmission delays. In case of delays the applications can agree on using another data format to reduce the needed bandwith.
The presented design decisions improve the efficiency but not cause trade-offs regarding maintainability and security. Table 6.2 summarizes the main design decisions affecting the scenario E2 in relation to possible risks.

Design Decisions	Risks
audio codecs supporting different sizes of bitstreams	quality of speech suffers from lower bitrates
video codecs which compress data in real-time	video codecs are optimized for low bitrate and low motion
RTP on top of UDP	packet loss RTP does not provide any mechanism to ensure timely delivery RTP does not provide quality-of-service guarantees

Table 6.2: ATAM output for the scenario E2

Scenario E3

The third efficiency scenario requires control mechanisms for connection parameters like jitter and bandwith. The connection parameters have to be checked from time to time to ensure a fast transmission with as low delay as possible. This task is fulfilled by the control channel manager. Every time the control loop is processed the control thread calculates the jitter for every active RTP channel. If the jitter is too high the jitter buffer is increased so that more data is buffered. The jitter buffer is used to write more packets at the same time with a little delay to the playing device, so the transmission delay is balanced and not so noticeable for the user. The connection and channel class shown in the logical view in Figure 5.16 provide operations for getting and setting the bandwith of singular RTP channels or the whole connection. So the endpoint manager has the possibility to adjust the bandwith according to available capacity. Table 6.3 summarizes the main desing decisions, which affect the efficiency scenario E3, in relation to trade-offs and risks caused by those design decisions.

DESIGN DECISIONS	TRADE-OFFS	RISKS
control channel manager calculate jitter and used bandwith	efficiency(-)	control channel manager has to access the share memory (bottleneck) for calculation
jitter buffer	efficiency(-)	jitter buffer adds delay to balance transmission delay

Table 6.3: ATAM output for the efficiency scenarios E3

Maintainability Scenarios

Scenario M2

The scenario M1, as shown in the utility tree in Figure 6.1, demands extensibility regarding new codecs, channel types and security mechanisms. This demand is satisfied by the architecture through the object oriented concept of inheritance. There are different abstract classes which form a kind of framework to enable the adding of own classes which implement new functionality. For example, the inheritance hierachy of the abstract channel class is shown in Figure 6.2. There are trade-offs between this abstract class pattern, which is described in [31], for increasing maintainability and efficiency and security. On the one hand the efficiency is decreasing at runtime because of the used polymorphism. On the other hand it is possible to integrate also security mechanisms which means an improvement of the system's security. The use of abstract classes has also a disadvantage towards maintainability because changes and extensions are only possible, without effecting the existing structure, if an abstract class for the further or changed class exists. System capabilities like codecs can only be registered at the endpoint manager when the system starts, as shown in Figure 5.11. New channel types must be known at compile time because there are couplings between channel objects and other objects, especially the connection object (Figure 5.16). So it is not possible to add functionality to the running system. Extensions have to be made at compile time.

Scenario M2

Scenario M2 requires the change of existing codecs, channel types and security mechanisms. This scenario is similar to scenario M1 because the effort on change concerns only system components which are implemented by classes which inherite from abstract

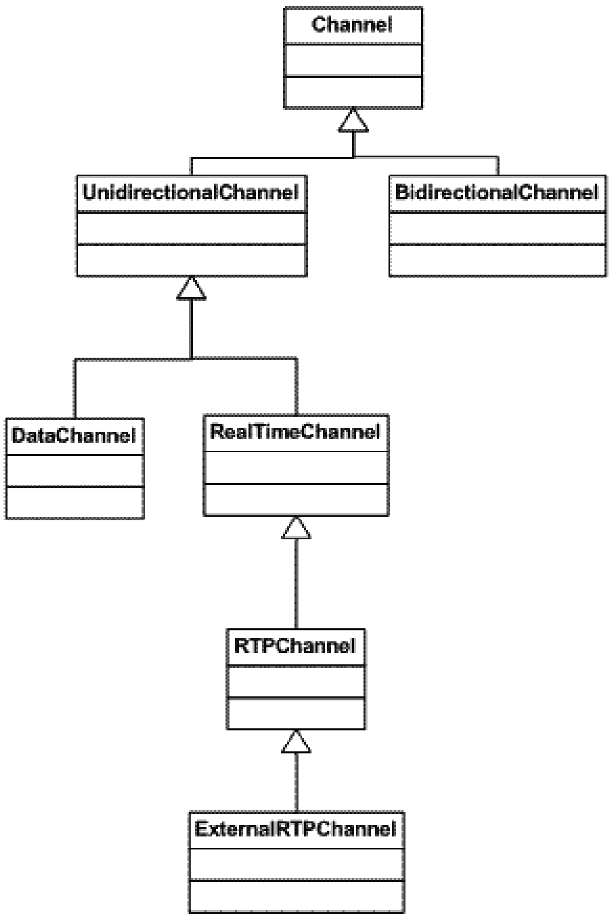


Figure 6.2: Inheritance hierachy of the abstract channel class

classes. So the abstract class pattern supports also this change scenario M2.

The trade-offs caused by this pattern are the same as in the above scenario M1. Also changes which are made must be known at compile time.

Scenario M3
The third maintainability scenario M3 regards changes to the signalling and/or control protocol. Although the architecture conceptual view consists of components implementing only a single task, the logical view does not support the functional decomposition strictly. So the behaviour of the signalling channel manager (Figure 5.11) and the control channel manager (Figure 5.11), which is shown in the Figures 5.15 and 5.14, is mapped to the logical view's connection class methods (Figure 5.16). Since this class also implements mainly the crucial functionality for the enabling a call, changes can effect the system's behaviour negatively. The connection class is also coupled strongly with almost every of the other classes. That means the architecture does not support this scenario because changes to the signalling and control protocol are not considered.

Table 6.4 summarizes the architectural designs decisions, trade-offs and risks for the maintainability scenarios.

Scenario	Design Decisions	Trade-offs	Risks

M1	abstract class pattern	maintainability(+)
		efficiency(-)
		security(+)
			after extension a new deployment of the system is necessary

M2	abstract class pattern	maintainability(+)
		efficiency(-)
		security(+)
			after a change a new deployment of the system is necessary

M3	- missing functional decomposition and strong coupling of connection class		- scenario is not supported - architecture does not regard such changes

Table 6.4: ATAM output for the maintainability scenarios

Security Scenarios

Scenario S1

The security scenario S1, which requires encryption of the transmitted data, is not realized by the architecture in the current state. Encryption is only used if the authentication mechanism for registration (Figure 5.7) at the gatekeeper is activated. The encryption is either based on Secure Sockets Layer (SSL) or Message Digest Algorithm 5 (MD5) hashing. Both mechanisms use at least 128 bit keys which means the decryption by a third party is very improbable. The algorithms can be changed and modified because the endpoint manager (Figure 5.11) uses an authentication agent which is implemented by an class which inheritates from the abstract class *H235Authenticators* implementing the authentication security.

Encryption is only supported for the registration, that means privacy and integrity is only assured for the exchanged authentication data. Which means the audio and video data is not encrypted and can be captured and analysed by a third party. The missing encryption affects positively the efficiency because encryption is a time consuming process, especially if the algorithms, which use matrix operations, are to be implemented. Audio and video data has to be processed fast because the maximum end-to-end delay should not be higher than 200 milliseconds ([32]). To avoid lowering the quality of the transferred media stream, the architecture does not support encryption of stream data.

Scenario S2

The second scenario S2, which demands authentication of the communication participants, is only supported for the case that a connection with the gatekeeper is established. The architecture supports authentication at the gatekeeper through the authentication agent which is used by the endpoint manager (Figure 5.11) of the caller and the one of the gatekeeper.
The architecture provides the abstract class *H235Authenticators* to enable the implementation of authentification mechanisms. The authentification data is encrypted, as mentioned in the previous scenario S1.

The authentification is by default not activated which improves the efficiency of the registration and admission since the time consuming authentication process is not executed. The abstract class pattern which is used for changing or extending the authentication component also improves the architecture's maintainability.

Architectural designs decisions, trade-offs, and risks for the security scenarios are summarized in Table 6.5.

Scenario	Design Decisions	Trade-offs	Risks

S1	-no special component for encryption
	- only authentication agent implements encryption
		efficiency(+)
		security(-)
			- scenario is not considered by architecture
			-privacy and integrity of transferred data is not assured

S2	authentication agent which is realised by the abstract class H235Authenticators	security(+)
		performance(-)
		maintainability(+)
			-authentication is only supported for the call scenario with gatekeeper
			-by default authentification is deactivated
			-efficiency decreases when authentication is used because of time consuming encryption/decryption

Table 6.5: ATAM output for the security scenarios

6.2 Evaluation using Architectural Metrics

Since the implementation of the OpenH.323 is available it is possible to collect architectural metrics from the code to support the evaluation of the architecture's maintainability. The two most interesting metics are the coupling and cohesion of the architecture's main classses, which metrics are described in Section 4.3.4. Further the complexity of the functions which implement the system's behaviour which is shown in the conceptual view in Figure 5.11, especially the signalling and control view, are interesting regarding the maintainability scenario M3.

The measurement is performed with the source code analyzing tool *understand for C++* [33]. This tool is offered as a free thirty day trial version. *Understand for C++* is able to measure the number of coupled classes and the lack of cohesion for a certain class in percents. High percentage corresponds with low cohesion. The following Table 6.6 shows the main classes of the logical view (5.16) with the coupling and cohesion metric.

Classes	Number of Coupled Classes	Percent Lack of Cohesion
Endpoint	43	98
Connection	91	97
Capability	11	88
Channel	12	87
Transport	11	86
Codec	7	86
H245Negotiator	2	81
H245NegLogicalChannels	8	71
Listener	4	88

Table 6.6: Coupling and cohesion metrics for the architecture's main classes

Furthermore, the tool *understand for C++* is utilized to determine the *cyclomatic complexity* (Section 4.3.4) of the main functions implementing the call initialization.

Table 6.7 gives an overview of the relation of the cyclomatic complexity and the related risk level.

Cyclomatic Complexity	Risk
1-10	a simple program, without much risk
11-20	more complex, moderate risk
21-50	complex, high risk program
greater than 50	untestable program (very high risk)

Table 6.7: Cyclomatic Complexity levels with related risk level [29]

The cyclomatic complexity of the methods, which are necessary for the implementing the signalling and control loop, are given in Table 6.8.

Class	Method	Cyclomatic Complexity
Connection	SendSignalSetup	39
	HandleSignallingChannel	10
	HandleSignalPDU	28
	WriteSignalPDU	4
	HandleControlChannel	7
	HandleControlPDU	5
	HandleControlData	4
Q931	Decode	8
	Encode	11

Table 6.8: Cyclomatic Complexity of essential methods

6.3 Cognitions from the Evaluation

In the two previous sections the architecture has been evaluated with the early evaluation approach ATAM, and the late evaluation approach architectural metrics. This section focuses on examining the collected outputs from both evaluations.

The ATAM evaluation has shown that the level of description of the architecture with the conceptual (Figure 5.11) and logical view (Figure 5.16) allows to assign the consideration of quality characteristics. Although ATAM does not evaluate the quality quantitively it helps to identify the lacks of quality and trade-offs among the three quality characteristics. The evaluation also showed that the OpenH.323 software is a typical Opensource project. Obviously, the OpenH.323 architecture focuses more on implementing the basic and necessary video conferencing functionality. The architecture consists of necessary components to perform a call but for example does not provide any additional features like the encryption of transmitted data because that is not essential to make a call. But the architecture, like it is typical for Opensource software, enables the addition of such features.

The quality characteristic efficiency is refined to three efficiency scenarios (Figure 6.1). The first efficiency scenario E1 aims especially on meeting the time limits demanded by the ITU standard for the protocol. Since these time limits are not really short, the implementations of the architecture should be able to meet them. Also the used design approaches as shown in Table 6.1 do not improve the architectures time behaviour. On the contrary, the time behaviour declines through them. Approaches like multithreading and shared memories are not used in a system which has to fulfil real-time requirements. The second and third efficiency scenario E2 and E3 (Figure 6.1) focus mainly on the quality of the transmitted audio and video data. To ensure fast processing and transmission, the architecture uses encoder and decoder which work in real-time and the RTP protocol as transmission protocol. Further, the architecture contains the control channel manager (Figure 5.11) which is observing the connection parameters like the jitter and used bandwith. The control channel manager (Figure 5.11) as well as the signalling channel manager are implementing the core functionality and these components are implemented in the logical view's connection class (Figure 5.16). Changes or extensions to that class are really risky because of high coupling and low cohesion as shown in Table 6.6. The methods of this class are very complex.

The H.323 protocol is very complex because it uses several other protocols, so the amount of exchange protocol directives is comparable high. There are voice over IP protocols whose degree of communication between components is lower which means that there are able to set up a call faster than H.323. Acccording to [30] the protocol architecture *The Session Initiation Protocol* (SIP) consits of less components and the amount of exchanged messages is also less. The complexity of the H.323 protocol is reflected also in OpenH.323 architecture. At runtime many objects are communicating and they exchange about 120 messages to initiate a call.

The efficiency evaluation with ATAM has shown the biggest weakness of the early evaluation and of early evaluation approaches like ATAM. Regarding the mentioned bottleneck, caused by the synchronisation of the shared memory, or the decreasing efficiency, caused by the context switches of scheduled threads, all these investigations are based on assumptions which are based on the expertise of the evaluatin team. There is no possibility to find quantive values with ATAM.

The utilization of object oriented techniques for implementing the OpenH.323 architecure aims on improving the quality characteristic maintainability. So for the system architect the maintainability is higher prioritize than the efficiency. The reason might be the moderate time limits for exchanging the protocol directives. The most critical efficiency requirement is ensuring a good quality of the audio and video stream. That this requirement is met by the architecture was stated above.

The quality characteristic maintainability is increased by providing a kind of framework to extend the architecture with further codecs, channel types and security mechanisms. The abstract class pattern is utilized to implement this framework. This causes of course trade-offs with the quality characteristic efficiency which is decreased by polymorphism. The collected design metrics *coupled classes* and *lack of cohesion* showed that the architecture is changeable regarding channel types and codecs. Table 6.6 shows that the coupling of the classes *Channel*, *Transport*, and *Codec* to other classes and their cohesion is relatively low in comparison to the classes *Endpoint* and *Connection*. The latter two classes contain the core functionality and it is quite difficult to change or extend them. This is shown by the ATAM output in Table 6.4. It was possible to recognize with ATAM from the logical view (Figure 5.16) that the connection class lacks a functional decomposition and is strongly coupled to other classes. Measuring the coupling and cohesion proved quantitatively that this class decreases the maintainability. The metrics are given in table 6.6. The high lack of cohesion value shows that the class could be decomposed in several more. Furthermore, measuring the complexity of the methods of the *Connection* and *Endpoint* class showed that there are complex methods which decrease the maintainability and especially the testability of the architecture. Another disadvantage of the architecture regarding maintainability is that changes and/or extensions cannot be performed on the running system. That means they must be known at compile time and the whole system must be deployed every time. Even for registering a new codec this effort is necessary.

In general, the architecture fulfils the stakeholders' demands for maintainability quite well. The maintainability profits mostly from the used object oriented paradigms and patterns, e.g. the abstract class patter, but these techniques affect the efficiency negatively. Only the core functionality of the system is difficult to change and/or extend.

The security demandings of the stakeholders are fulfiled partly by the architecture. The second security scenario S2 (Figure 6.1) is fulfilled by the authentication and authorization mechanisms used for registration and admission at the gatekeeper. The user's authentication data is transmitted encrypted. But the communication itself is not secure. But additional security features can be implemented because the architecture provides the means to implement further encryption algorithms and secure channels.
The missing security features, e.g. encryption of transmitted data, benefit the system's efficiency.

7 Conclusion

The thesis' primary objective is the early evaluation of the OpenH.323 protocol software architecture. The evaluation is performed with two architecture evaluation methods as described in Section 4.3. The Architecture Trade-Off Analysis Method (ATAM) and the architectural metrics approach are utilized. ATAM was described in Section 4.3.3 and architectural metrics in Section 4.3.4

For the architectural description a conceptual and logical view have been utilized and they are presented in Section 5.3.. The conceptual view gives a general overview of the functional components with their relations towards each other. The case study showed that the conceptual description is very suitable for an ATAM-based evaluation. Even from this high-level description it is possible to identify risk and mistakes in the planned system. Depending on the expertise of the evaluation team, even assumptions regarding runtime behaviour are possible.

The logical view is an object-oriented decomposition of the software system. The decomposition is achieved by abstracting from the functionality of the conceptual view's components. The logical view is very suitable to assess extendability and changeability of the software system because it enables the collecting of architectural metrics to determine coupling and cohesion.

The performed evaluation regarding the three identified quality characteristics proved that the used architectural description is a useful input for an early evaluation like ATAM because it was possible to identify quality weaknesses as well as points of trade-off. Actually, the evaluation showed that the OpenH.323 architecture focuses mainly on the necessary functionality to enable video conferencing. Regarding the three quality characteristics, the analysis of the evaluation results in Section 6.3 points out that the design of the architecture mainly focuses on the maintainability characteristic. The efficiency of the OpenH.323 protocol is essential for the processing and transmitting of audio and video data with a low end-to-end delay. That is why the architecture implement mechansims like real-time encoding/decoding, jitter buffer, and RTP. But since the time limits for exchanging H.323 protocol directives for signalling and status control are quite moderate as shown in the usage scenarios in Section 5.2.2, the architecture utilizes design decisions which even decrease the efficiency. The security requirements are only partly implemented but it is possible to extend the architecture with security features.

ATAM's biggest disadvantage is the absence of metrics which give quantitive evidence about the degree of fulfilment of certain quality characteristics. The whole ATAM assignment is based on the expertise of the evaluation team. ATAM only enables assumptions regarding the system's behaviour and the degree of fulfilment of certain quality characteristics, like efficiency or security, at runtime. So the quality of the ATAM evaluation or the early evaluation in general depends on the expertise of the evaluation team and the quality of the architectural description as input for the evaluation.

This shows that it is possible to recognize conceptual flaws by performing a software architecture evaluation by just using the ATAM evaluation is not enough to assign the quality. Especially characteristics which occur at runtime like efficiency must be evaluated through mathematical models or simulations on the architectural level to make correct predictions. Just relying on expertise is not enough for a sound evaluation. There is definitely a need for further methods and approaches to evaluate software architecture regarding quality characteristics which occur at runtime already on the architectural level. Also the development of tools which recognize conceptual mistakes and points of trade-off in the architectural description are needed to avoid mistakes in the early development stages.
The existing tools collecting architectural metrics from the source code description only enable the maintainability evaluation.

8 References

[1] T. Kärkkäinen, N. Hämäläinen, J. Ahonen. Why to evaluate enterprise and software architectures: objectives and use cases. Information Technology Research Institute, University of Jyväskylä. Jyväskylä, Finland, 2005.

[2] Rick Kazman, Len Bass, Paul Clements. *Software Architecture in Practice.* Addison Wesley, 2 edition, April 2003.

[3] Helmut Balzert. *Lehrbuch der Software-Technik.* Spektrum Akademischer Verlag, 2 edition, 2001.

[4] B. Groehne P. Tabeling A, Knoepfel. *Fundamental Modeling Concepts: Effective Communication of IT Systems.* Wiley and Sons Ltd, March 2006.

[5] AISA project - quality management of enterprise and software architectures. `http://www.titu.jyu.fi/aisa/index.htm`, October 2006.

[6] M. Shereshevsky, H. Ammari. Information theoretic metrics for software architectures. Chicago, IL, USA, 2001. IEEE Computer Society. Proceedings of the 25th Annual Internation Computer Software and Application Conference (COMPSAC 2001).

[7] IEEE standard recommended practice for architecture description. IEEE Std 1471-2000.

[8] IEEE standard for a software quality metrics methodology. IEEE Std 1061-1998, December 1998.

[9] P.B. Kruchten. The 4+1 view model of architecture. *IEEE Software*, 12(6):42–50, November 1995.

[10] C. Hofmeister, D. Soni, R. L. Nord. Software architecture in industrial applications. Proceedings of the 17th international conference on Software engineering. pages 196–207, Seattle, Washington, USA, 1995.

[11] William E. Burr, Samuel H. Fuller. Measurement and evaluation of alternative computer architectures. *Computer*, 10:24–35, October 1977.

[12] M. Lipow, B. W. Boehm, J. R. Brown. Quantitative evaluation of software quality. San Francisco, California, United State, 1976. IEEE Computer Society Press. Proceedings of the 2nd international conference on Software engineering.

[13] M. Barbacci, M. H. Klein. Quality attributes. Technical report, 1995.

[14] Karl E. Wiegers. *Software Requirements 2: Practical techniques for gathering and managing requirements throughout the product development cycle.* Microsoft Press, 2 edition, 2003.

[15] F. Lasavio, L. Chirinos. Iso quality standards for measuring architectures. *The Journal of Systems and Software*, 72:209–223, 2004.

[16] Mark Klein, Paul Clements, Rick Kazman. *Evaluating Software Architectures: Methods and Case Studies.* Addison Wesley, 1 edition, October 2001.

[17] M. Svahnberg. An industrial study on building consensus around software architectures and quality attributes. *Information and Software Technology*, 46:805–818, 2004.

[18] M. Lindvall, R. Tesoriero. Avoiding architectural degeneration: An evaluation process for softwareg. pages 77–86. IEEE Computer Society, 2002. Proceedings of the Eighth IEEE Symposium on Software Metrics (METRICS'02).

[19] A. V. Corry, J. Bardram, H. B. Christensen, M. Ingstrup, and K. M. Hansen. Exploring quality attributes using architectural prototyping. pages 155–170. Springer-Verlag Berlin Heidelberg, 2005. Proceedings of the First International Conference on the Quality of Software Architectures (QoSA 2005).

[20] Paul E. Jones. Overview of H.323. `http://www.packetizer.com/voip/h323/papers/overview_of_h323_files/frame.html`, June 2004.

[21] H. Astudillo. Five ontological levels to describe and evaluate software architectures. *Rev. Fac. Ing. - Univ. Tarapacá*, 13:69–76, 2005.

[22] S. Ferber. Reviewing Software Architecture: Experience in Applying ATAM at Bosch, October 2002. Robert Bosch GmbH, Frankfurt, Germany.

[23] M. Denford, K. Dunsire, T. O'Neill. The abacus architectural approach to computer-based system and enterprise evolution. pages 62–69, April 2005. 12th IEEE International Conference and Workshops on the on Engineering of Computer-Based Systems.

[24] J. Bosch. Software architecture assessment. International Summer School on Usability-Driven Software Architecture. Tampere, Finland, 2005. University of Technology.

[25] P. Molin, J. Bosch. Software architecture design: evaluation and transformation. pages 7–12, March 1999. IEEE Conference and Workshop on Engineering of Computer-Based Systems.

[26] H. Grahn, M. Mattsson, F. Mårtensson. An approach for performance evaluation of software architectures using prototyping. USA, 2003. Proceedings of the 7th IASTED International Conference on Software Engineering and Applications.

[27] W. Diestelkamp L. Lundberg, D. Häggander. Conflicts and trade-offs between software performance and maintainability. volume 2047, pages 56–67. Springer-Verlag, 2001.

[28] Dominik Sacher. Diagrams on openh323 functions. `http://www.openh323.org/docs/diagrams.html`.

[29] T. J. McCabe. A complexity measurement. pages 308–320, December 1976. IEEE Transactions on Software Engineering, 2.

[30] S. Afsharian M. Castaldi, P. Inverardi. A case study in performance, modifiability and extensibility analysis of a telecommunication system software architecture. pages 281–290, 2002. Proceedings. 10th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems.

[31] Bobby Woolf. The abstract class pattern. 1997. In PLOP Proceedings.

[32] W. Mandranwa, P. Calyam, M. Sridharan, P. Schopis. Performance measurement and analysis of h.323 traffic. Antibes Juan-les-Pins, France, 2004. In Proceedings of the 5th International Workshop on Passive and Active Network Measurement, PAM 2004.

[33] Scientific Toolworks Inc. Understand for C++. `http://www.scitools.com/products/understand/cpp/product.php`.

Glossary

End-to-end delay *End-to-end delay* includes compression and transmission delay at the sender, the propagation, processing, and queuing delay in the network, and buffering and decompression delay at the receiver.

FMC The *Fundamental Modeling Concepts* primarily provide a framework for the comprehensive description of software-intensive systems. It is based on a precise terminology and supported by a graphical notation which can be easily understood.

ICT Information and Communication Technology

IEC The *International Electrotechnical Commission* is responsible for standardization of electrical equipment.

IEEE The *Institute of Electrical and Electronics Engineers* is an international non-profit, professional organization for the advancement of technology related to electricity.

ISDN The *Integrated Services Digital Network* is a type of circuit switched telephone network system, designed to allow digital transmission of voice and data over ordinary telephone copper wires, resulting in better quality and higher speeds than available with analog systems.

ISO The *International Organization for Standardization* is an international standard-setting body composed of representatives from national standards bodies.

ITU The *International Telecommunication Union* is an international organization established to standardize and regulate international radio and telecommunications.

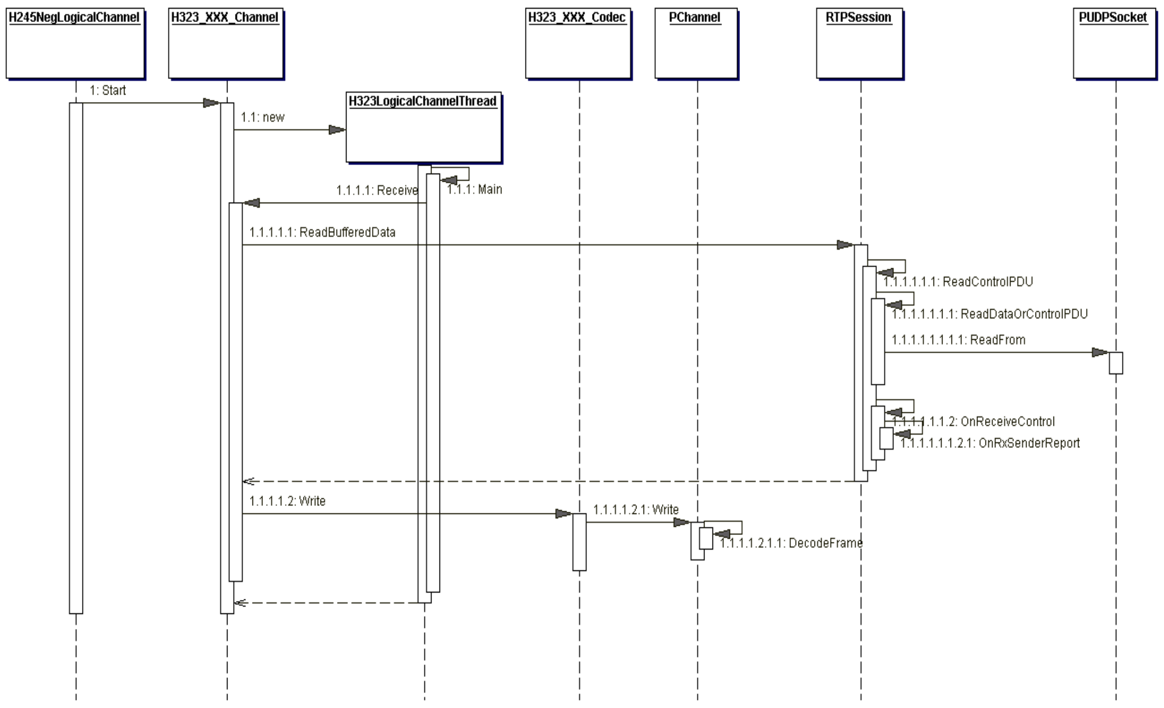
MCU Multipoint Control Unit is a device commonly used to bridge videoconferencing connections. The Multipoint Control Unit is an endpoint on the LAN which provides the capability for three or more terminals and gateways to participate in a multipoint conference.

Multithreading *Multithreading* means the concurrent execution more than one program by the same machine.

Mutex The *Mutual exclusion mechanism* is a means for inter task communication usually provided by the operating system. Concurrent thread use a mutex to ensure that only one thread can access a shared resource. Any other thread who tries to get the mutex will either be blocked until the mutex is released, or its request will be rejected.

UML The Unified Modeling Language is a non-proprietary object modeling and specification language used in software engineering. UML is a general-purpose modeling language that includes a standardized graphical notation that may be used to create an abstract model of a system, sometimes referred to as the UML model.

A Additional Diagrams



H245NegLogicalChannel
H323_XXX_Channel
H323_XXX_Codec
PChannel
RTPSession
PUDPSocket
1: Start
H323LogicalChannelThread
1.1: new
1.1.1.1: Receive
1.1.1: Main
1.1.1.1.1: ReadBufferedData
1.1.1.1.1.1: ReadControlPDU
1.1.1.1.1.1.1: ReadDataOrControlPDU
1.1.1.1.1.1.1.1: ReadFrom
1.1.1.1.1.2: OnReceiveControl
1.1.1.1.1.2.1: OnRxSenderReport
1.1.1.1.2: Write
1.1.1.1.2.1: Write
1.1.1.1.2.1.1: DecodeFrame

Figure A.1: Communication between objects for starting data reception [28]

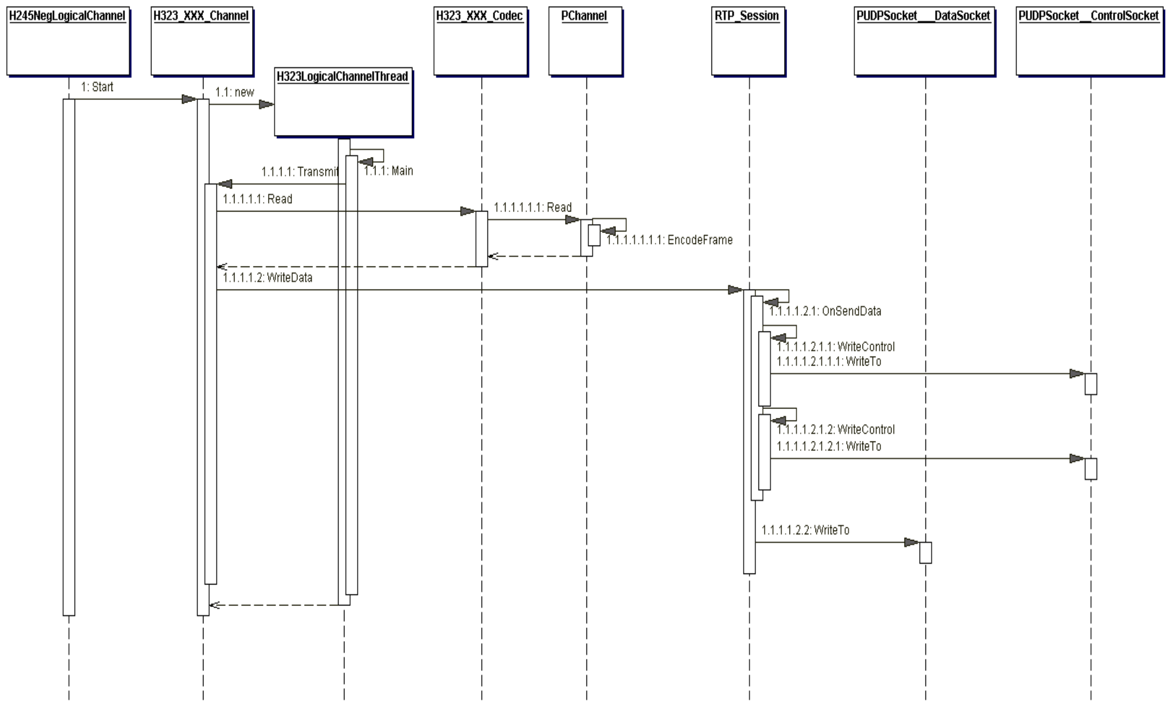


Figure A.2: Communication between objects for starting data transmission [28]

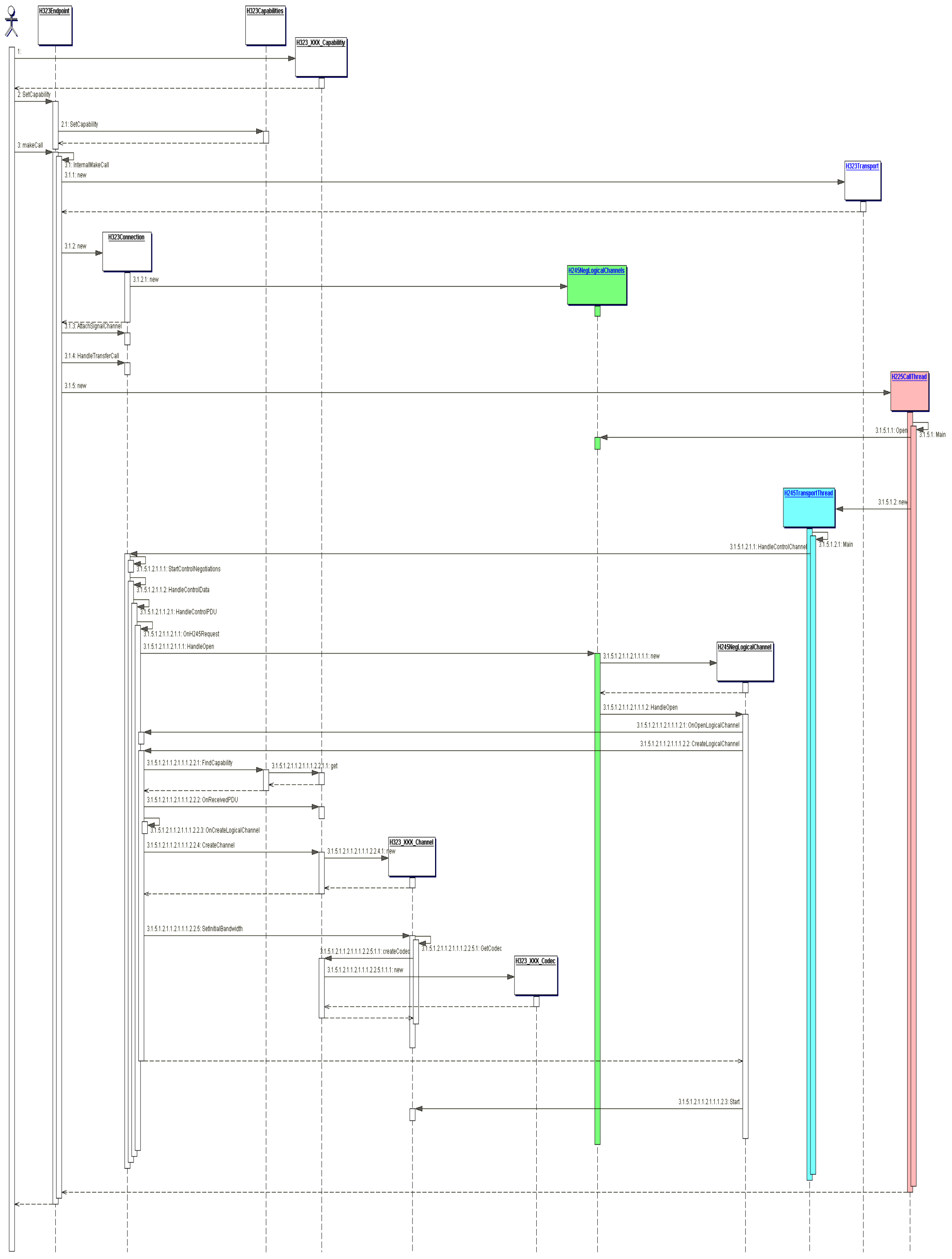

H323Endpoint
H323Capabilities
H323_XXX_Capabilitiy
1:
2: SetCapability
2.1: SetCapability
3: makeCall
3.1: InternalMakeCall
3.1.1: new
H323Transport
H323Connection
3.1.2: new
3.1.2.1: new
H245NegLogicalChannels
3.1.3: AttachSignalChannel
3.1.4: HandleTransferCall
3.1.5: new
H225CallThread
3.1.5.1.1: Open
3.1.5.1: Main
H245TransportThread
3.1.5.1.2: new
3.1.5.1.2.1.1: HandleControlChannel
3.1.5.1.2.1: Main
3.1.5.1.2.1.1.1: StartControlNegotiations
3.1.5.1.2.1.1.2: HandleControlData
3.1.5.1.2.1.1.2.1: HandleControlPDU
3.1.5.1.2.1.1.2.1.1: OnH245Request
3.1.5.1.2.1.1.2.1.1.1: HandleOpen
3.1.5.1.2.1.1.2.1.1.1.1: new
H245NegLogicalChannel
3.1.5.1.2.1.1.2.1.1.1.2: HandleOpen
3.1.5.1.2.1.1.2.1.1.1.2.1: OnOpenLogicalChannel
3.1.5.1.2.1.1.2.1.1.1.2.2: CreateLogicalChannel
3.1.5.1.2.1.1.2.1.1.1.2.2.1: FindCapability
3.1.5.1.2.1.1.2.1.1.1.2.2.1.1: get
3.1.5.1.2.1.1.2.1.1.1.2.2.2: OnReceivedPDU
3.1.5.1.2.1.1.2.1.1.1.2.2.3: OnCreateLogicalChannel
3.1.5.1.2.1.1.2.1.1.1.2.2.4: CreateChannel
3.1.5.1.2.1.1.2.1.1.1.2.2.4.1: new
H323_XXX_Channel
3.1.5.1.2.1.1.2.1.1.1.2.2.5: SetInitialBandwidth
3.1.5.1.2.1.1.2.1.1.1.2.2.5.1.1: createCodec
3.1.5.1.2.1.1.2.1.1.1.2.2.5.1: GetCodec
3.1.5.1.2.1.1.2.1.1.1.2.2.5.1.1.1: new
H323_XXX_Codec
3.1.5.1.2.1.1.2.1.1.1.2.3: Start

Figure A.3: Messages for call initiation [28]

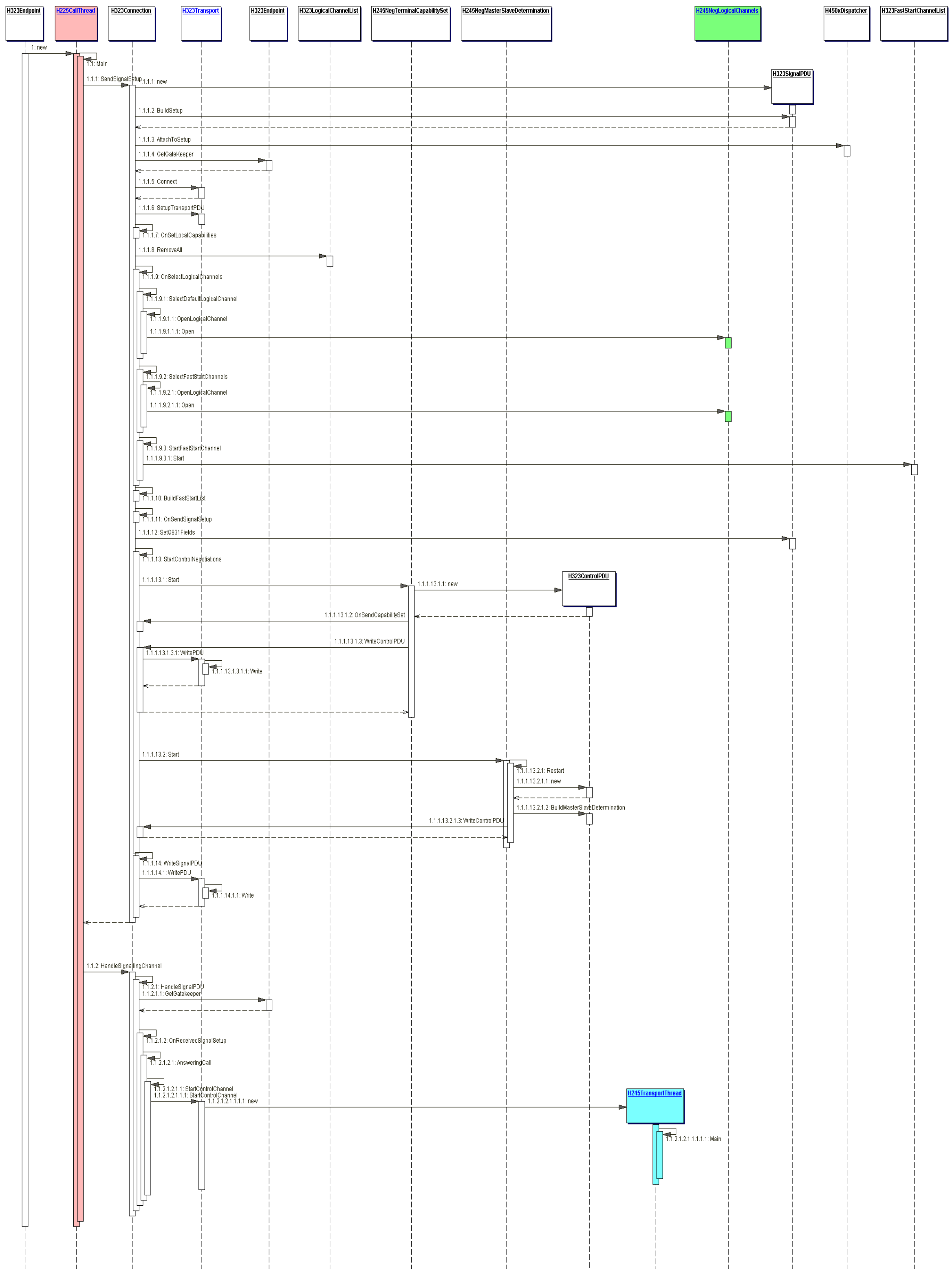

H323Endpoint
H225CallThread
H323Connection
H323Transport
H323Endpoint
H323LogicalChannelList
H245NegTerminalCapabilitySet
H245NegMasterSlaveDetermination
H245NegLogicalChannels
H450xDispatcher
H323FastStartChannelList
H323SignalPDU
H323ControlPDU
H245TransportThread
1: new
1.1: Main
1.1.1: SendSignalSetup
1.1.1.1: new
1.1.1.2: BuildSetup
1.1.1.3: AttachToSetup
1.1.1.4: GetGateKeeper
1.1.1.5: Connect
1.1.1.6: SetupTransportPDU
1.1.1.7: OnSetLocalCapabilities
1.1.1.8: RemoveAll
1.1.1.9: OnSelectLogicalChannels
1.1.1.9.1: SelectDefaultLogicalChannel
1.1.1.9.1.1: OpenLogicalChannel
1.1.1.9.1.1.1: Open
1.1.1.9.2: SelectFastStartChannels
1.1.1.9.2.1: OpenLogicalChannel
1.1.1.9.2.1.1: Open
1.1.1.9.3: StartFastStartChannel
1.1.1.9.3.1: Start
1.1.1.10: BuildFastStartList
1.1.1.11: OnSendSignalSetup
1.1.1.12: SetQ931Fields
1.1.1.13: StartControlNegotiations
1.1.1.13.1: Start
1.1.1.13.1.1: new
1.1.1.13.1.2: OnSendCapabilitySet
1.1.1.13.1.3: WriteControlPDU
1.1.1.13.1.3.1: WritePDU
1.1.1.13.1.3.1.1: Write
1.1.1.13.2: Start
1.1.1.13.2.1: Restart
1.1.1.13.2.1.1: new
1.1.1.13.2.1.2: BuildMasterSlaveDetermination
1.1.1.13.2.1.3: WriteControlPDU
1.1.1.14: WriteSignalPDU
1.1.1.14.1: WritePDU
1.1.1.14.1.1: Write
1.1.2: HandleSignallingChannel
1.1.2.1: HandleSignalPDU
1.1.2.1.1: GetGatekeeper
1.1.2.1.2: OnReceivedSignalSetup
1.1.2.1.2.1: AnsweringCall
1.1.2.1.2.1.1: StartControlChannel
1.1.2.1.2.1.1.1: StartControlChannel
1.1.2.1.2.1.1.1.1: new
1.1.2.1.2.1.1.1.1.1: Main

Figure A.4: H225CallThread messages for call setup [28]

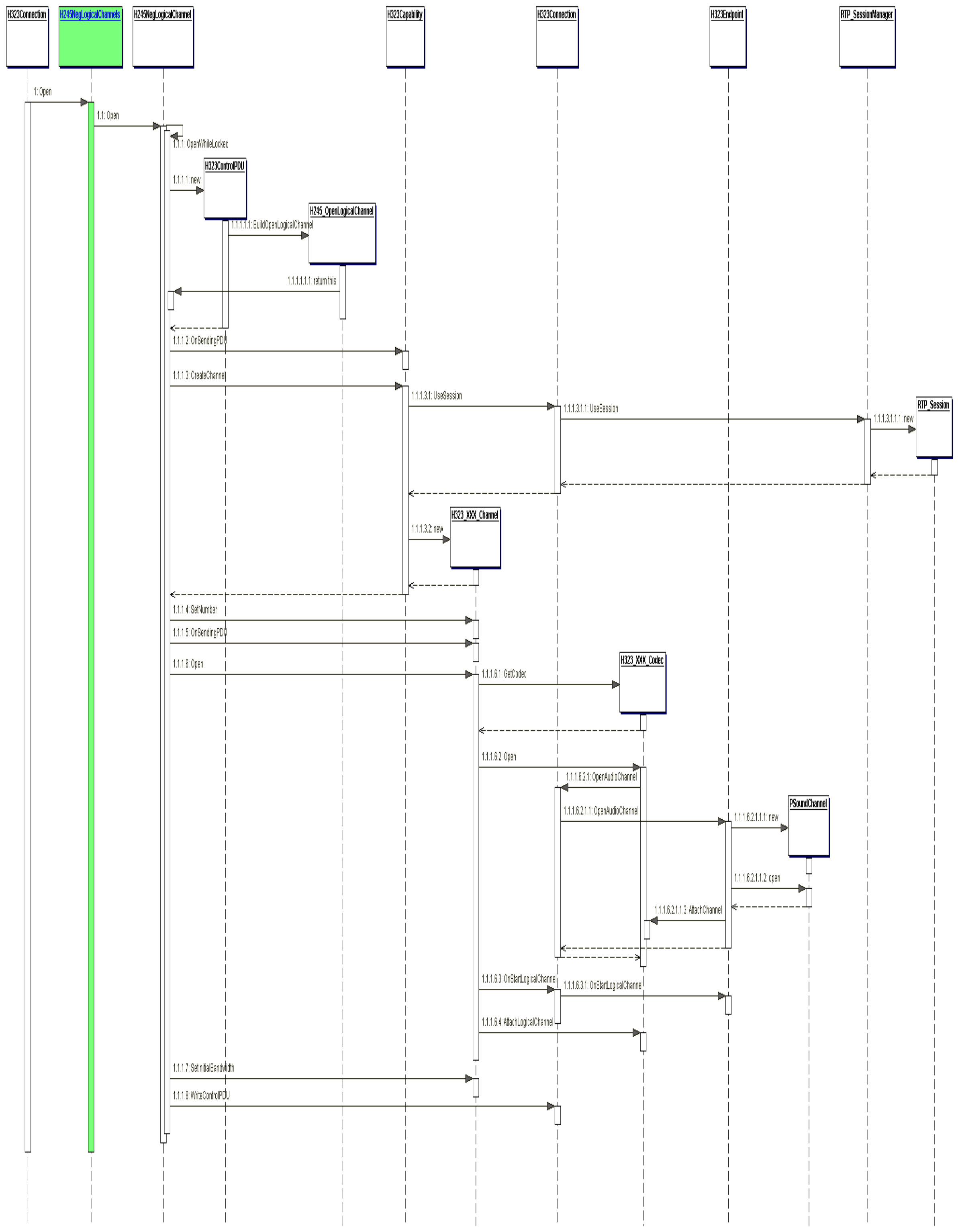


Figure A.5: Communication between objects for opening data channel [28]