# Implementing QoS Aware Component-Based Applications

Avraam Chimaris and George A. Papadopoulos

Department of Computer Science, University of Cyprus
75 Kallipoleos Street, POB 20537, CY-1678, Nicosia, Cyprus
chimaris@cytanet.com.cy, george@cs.ucy.ac.cy

**Abstract.** By QoS (Quality of Service), we often refer to a set of quality requirements on the collective behavior of one or more objects. These requirements enable the provision of better service to certain data flows. The developer can either increase the priority of a data flow or limit the priority of another data flow, in order to tune the proper "parameters" that support quality requirements. Nowadays, the use of contracts for software components is a novel way of guaranteeing their quality. It is a rather cumbersome task to design components that comply with contracts, because different problem dimensions or quality aspects have to be addressed at the same time. In this paper, we employ a simple methodology to show how we can design and develop QoS components that are based on Aspect-Oriented Programming and Model Transformation. We use a Tele-Medicine framework to show how we can embed to the final product a set of QoS contracts. We implement two such contracts that support QoS in communication and teleconferencing. We describe all the steps of the analysis, design and implementation in order to denote the advantages of using this novel way of *weaving* quality contracts into QoS applications.

## 1 Introduction

The development of QoS components based on contracts ([9]) is a novel way in software analysis and design. Modern methodologies like Aspect Oriented Programming (AOP) ([5]) focus on non-functional requirements that are usually difficult to identify. By dividing the problem into several simpler problems, we can concentrate easily on the major aspects of an application that require QoS control. With AOP, partial implementations can be developed, each one addressing a single quality aspect of the component. These partial implementations can be woven together by specially designed aspect weavers to form a complete component complying with a certain contract. The methodology that we are using follows the MDA approach ([6, 7]) whereby the system functionality specification can be separated from its implementation on any specific technology platform, the system's architecture being initially language, vendor and middleware neutral. For creating MDA-based applications, the first step is to create a Platform Independent Model (PIM), which can be expressed in UML ([11]). Such a PIM is easily mapped onto a Platform Specific Model (PSM) to a specific target platform. MDA is a way to

R. Meersman, Z. Tari (Eds.): CoopIS/DOA/ODBASE 2004, LNCS 3291, pp. 1173–1189, 2004.

separate an application's architecture from its implementation. Consequently, we get the advantage of the previous separation of the system design (PIM and PSM) but also the implementation is separated into two phases. In the first phase, only the functionality of the system is described along with its required extra-functional properties (the PIM model). In this phase, we avoid including any platform-dependent coding. In the second phase, platform specific design decisions are introduced in the form of "aspects", specific pieces of design that are needed to realize the extra-functional properties of the software components (this is related to the PSM model). When modeling the PIM, the functionality is described by means of the usual UML constructs. In order to describe the extra-functional properties, UML is extended to support contract types. Through this extension, it is possible to model "contracts" and the concept of "aspect". This modeling has special features that have to be considered and the implementation of the final product is much more difficult to be handled because the route from the model to the final coding is complicated. The QCCS methodology that has been used in this work, involves the use of a tool that not only supports these new extensions of the UML models but it also includes a transformation process that helps the designer to transform PIM models into PSM models which include the required aspects. The PSM models are then used to generate the code of the new application. This new application includes the contracts weaved into the final version of the code generated.

In this paper, we will illustrate the advantages of the QCCS methodology by presenting the steps to be followed in the implementation of a contract-aware component-based application. This application, a Tele-Medicine framework, supports the remote monitoring of patients taking medication from doctors. This framework was initially implemented without supporting any QoS aspects. We will use a *Latency* contract and a *Bandwidth* contract to enhance the initial system by including these required aspects. These two contracts can guarantee that the system supports the existence of communication channels and an acceptable bandwidth rate during teleconference sessions. We will not only use the contracts in a high level analysis but we will also present in detail all the steps that will be followed in the design and implementation of this framework that are related to the MDA approach and model transformation. The rest of the paper is structured as follows: Section 2 provides a brief overview of the QCCS methodology. Section 3 analyzes the Tele-Medicine framework that is used as an example of applying our methodology. Section 4 analyzes the application for QoS, identifies certain related requirements, and then presents the steps all the way to code generation, after the weaving of the contracts into the final product. Finally, section 5 outlines some concluding remarks and future work.

## 2 Designing QoS Applications: The QCCS Approach

QCCS (Quality Controlled Component-based Software, [12]) was an IST project sponsored by the European Commission that developed a methodology and supporting tools for the creation of components based on contracts and AOP. Components that have been designed according to the QCCS methodology have

proven properties, which are formally specified in contracts and can therefore be safely applied to build complex systems and services. The QCCS methodology complemented other existing methodologies and enhanced them. In particular, the methodology focuses on non-functional issues for the specification part and on aspect weaving and transformation for the design side. QCCS uses extensively software architecture models to support its methodology of model weaving and transformation. These models are based on extensions of the UML metamodel ([8]). UML ([11]) was chosen as the QCCS standard modeling language because of its widespread use in the industry, its extensibility properties and the strong growth of transformation tools in academic research as well as in industrial tool companies. While UML is a powerful and widespread modeling language, it is nothing more than a notation used within a software development process. Building such a process for quality controlled component construction was precisely one of the main objectives of the QCCS project and associated methodological issues were carefully examined.

The UML metamodel was used in the analysis phase, in which defined meta-classes denoted the required contracts, themselves denoting the non-functional requirement aspects. By using the MDA approach, the above notation was used in the construction of the PIM models. The model transformation technique was used afterwards to transform the PIM model into PSM models by applying certain transformation rules specific to the selected contracts.

In order to be more specific on how the methodology is used we are presenting the following simple steps for the end-users that need to develop a QoS-aware component-based application. These steps are the following ones:

**Identify contracts:** The first step on producing a quality-aware component-based application is to identify the quality-related requirements which will be implemented as contracts. Examples of such requirements for a quality-aware component are, say, the maximum timeout for an execution or the minimum network bandwidth for a video-conferencing transmission, etc.

**Design the application in UML:** The software developer is then called to design the application (a Platform Independent Model) using a CASE tool that was developed as part of the QCCS framework and supports the development of contracts. There is no limitation to what kind of contracts may be implemented but the contracts should be implemented before the developer starts working on the application design. That is why a more specialized developer (*Contract Developer*) is called to implement the required contracts and weavers. The QCCS methodology is suggesting the use of such specialists that can implement reusable contract weavers supporting QoS aspects. The *Contract Developer* designs the transformation models that transform and inject the QoS code into the final application. The above mentioned tool assists the developer in transforming the PIM model to a PSM (Platform Specific Model) one by using the already made contract weavers. This PSM model is the initial weaved model that contains the QoS contracts.

**Code generation:** The CASE tool then takes the PSM model and produces code for the architecture. The code generated includes the project files in the selected platform (for example .NET), the structure code for the different components and, most importantly, the code for the QoS contracts.

Therefore, a developer following the above steps can easily develop a quality-aware application that supports a set of quality contracts. In the following sections,

we will analyze this approach by using the Tele-Medicine framework and define transformation rules that embed the QoS aspects into the final product. The transformation rules that will presented can easily be used to transform and weave any similar QoS application. The developer can use in the PIM model the same contracts as we use them to generate code that assists the QoS checking.

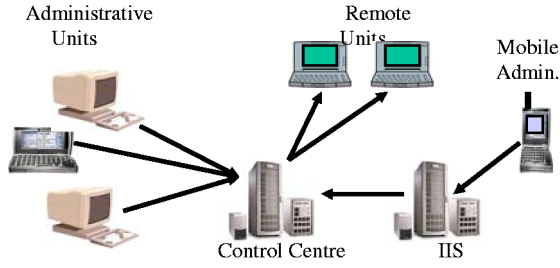


**Fig. 1.** The Tele-Medicine Framework

## 3 A Tele-Medicine Framework

The Tele-Medicine framework supports remote monitoring of medication devices in patients' homes from remotely connected doctors. The medication plan can be remotely modified, and checks can be made to assure that the patient is taking it on time without long delays. The framework also supports teleconferencing between doctors and patients in order to discuss any issues related to the treatment of the latter.

### 3.1 Analysis of the Tele-Medicine Framework

Initially, the Tele-Medicine framework was analyzed, designed and developed without the support of any QoS contracts. However, some main "contracts" of parameter checking or pre-condition states where included as parts of a standard design and implementation phase. Even if the system was not QoS aware, it was structured into four component-based subsystems ([1]) (Fig. 1 shows the overall architecture):

– The **Remote Units** are the units handling a patient's medication. Each of the remote units is serving a patient, and has specific information about his required medication and the exact time on which he has to take his medication. By using this information, the remote units alert the users to take their medication on time and in case the patients neglect to do so, a message is sent to a control centre in order to alert a doctor. The remote units also keep track of the patients' medicine stock, in order to pre-order medicines before they run out. This unit can also support teleconference communication with the doctors by using a media player and a media encoder. Even though the teleconference and the connectivity issues were crucial in the system, this initial design did not include any "active" monitors that could alert on certain failures. Some elementary quality control checking for specific tasks was included but any notifications regarding degradation of performance were missing.

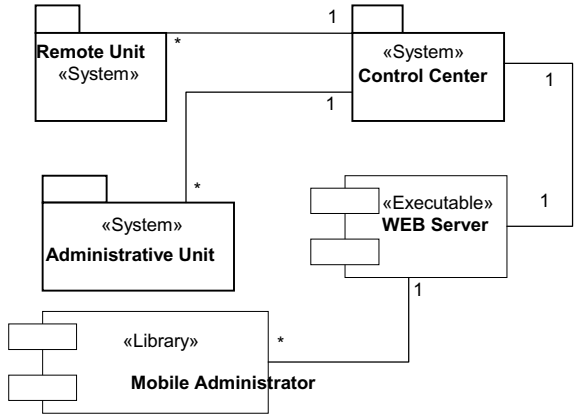


**Fig. 2.** The Tele-Medicine Components

- The **Administrative Units** are the units controlled by the doctors. Through them, the doctors have the ability to monitor their patients' status (notably if they have taken their medication) and send messages to them. Also, they can teleconference with them and alter their medication plan or the dose dynamically.
- The **Mobile Administrator** is a web interface for the administrative units. This interface, is essentially the same with that of the administrative units.
- The **Control Centre** is the most vital unit of the framework. Running on a stable and secured machine, the control centre is aware of and coordinates the whole system. It receives any messages from the units and forwards them to the proper receiver (doctor's administrative unit or patient's remote unit respectively).

Figure 2 shows the above described main system components. In this section, we will examine the structure of these components and then we will see how the QoS issues helped us to implement the QoS contracts. Below we will analyze the Tele-Medicine framework from the point of view of communication and teleconference, in order to determine how these features were initially implemented (without QoS) and then how they were enhanced with QoS aspects. In this section, we will analyze only those components that are involved in these aspects. In section 4 we will combine this analysis with the contract analysis that brings in QoS requirements.

**Communication Issues.** In order to implement communication between the units, it was necessary to create a communication "protocol" which could cover the needs on communication issues. Therefore, a Client/Server communication approach was adopted ([2]), using TCP channels and specific components. These communication components are used in the lower level of communication and therefore an infrastructure was needed to handle both channels and messages transmissions and receipts ([3, 4]). This infrastructure needed to satisfy the following requirements: create and destroy communication channels, coverage of all "communication scenarios" between the Tele-Medicine units, fast data structure analysis of the received packages. These requirements were included in the implementation of highly efficient components that handle the communication needs ([5, 6]). An examination of the above requirements reveals as the most important

feature regarding QoS needs in communication aspects to be the first one, which is involved in channels administration. We will therefore analyze the involved communication components in the communication aspect. The communication aspect is similar in all the Tele-Medicine components, which use a *Communicator* class for handling the communication channels. This class cooperates with some data classes that are used to handle the units' data. These classes are contained in the Control Unit, which is the most important sub-component of the major Units in the framework (Remote Units, Administrative Units and Control Center). These Control Units handle the communication (Communicator) and they process and store data messages. Below, we will present Class diagrams that denote the structure of the Tele-Medicine Units and how these components are used for supporting the required features.

**Teleconference Issues.** Here we will describe how this type of communication is achieved. We use two components that contain the necessary parts, technologies and protocols for such communications. Encoder is a component that is capable of encoding a given file or stream and transmitting it to a given port. So, by using a player in the other communication ending, we can show the transmitted stream. Each Unit contains an encoder and a media player for presenting the received stream. In the teleconference protocol, we use messages in order to set-up the teleconference components. These messages contain information about encoder ports, settings of transmission and request/accept dealings. Before we proceed with the analysis, we must mention that these two types of components are contained in a more generic component, the *Teleconference* component that is responsible for handling both the sending and the receiving of media stream. We mention this because afterwards this component will be chosen in the QoS enhancement.

Below we analyze each unit and describe how the units are functioning in a Tele-Medicine environment by emphasizing the parts that we will use in the QoS enhancement (communication and teleconference issues).

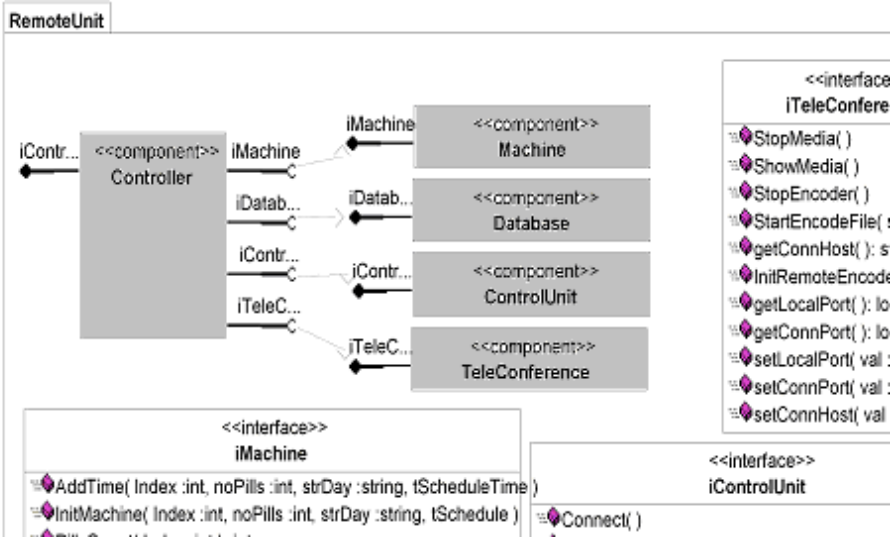


**Fig. 3.** Remote Unit Structure

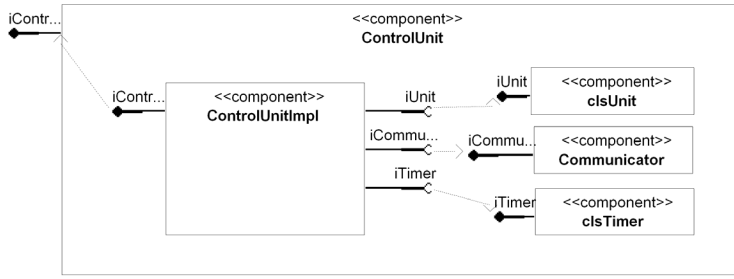


**Fig. 4.** Remote Unit – Control Unit

**Remote Units.** The Remote Units are handled by the Administrative Units in the Tele-Medicine framework. They have their data stored in a structured form and they have functionalities for loading, updating and communicating with other units. We have mentioned that Remote Units cannot be handled directly. They are connected with the Control Centre, which is responsible for retrieving and forwarding messages to the Remote Units. The Remote Units are using a data class for storing their data and a Communicator class for connecting to the Control Centre. This was mentioned in the communication issues section and it is presented in figures 3 and 4. The Controller Class (Control \Unit) handles both Communicator and data class. The interface of the Communicator is initializing the channels and the Control Unit is using a more abstract interface that is used to transmit messages (fig. 3). The Control Units are also used in the other telemedicine units; they have different interfaces but their role and functionality is quite the same. The role of the Control Units in the framework is summarized below:

- Handle the *Communicator* for connecting to the Control Centre.
- Send the initial data of a unit to the Control Centre after a successful connection.
- Perform the activities on messages arrivals. These messages are mainly updating, teleconferencing and synchronization messages.

In these diagrams, it is obvious that the Control Unit is using a communicator to handle the communication channel and a teleconference component to handle the teleconference communication.

**Control Centre.** The Control Centre is the most important unit in the Tele-Medicine framework. It is the middleware unit, the coordinator, the message handler, the very heart of the system. The Control Centre should normally run on a powerful machine and is using a web server in order to implement the Mobile Administrator role. The major components comprising the Control Centre unit are: a *Controlling Class*, *Communicator Switches* (to support the set of connected Units), *Coordinators* (to administrate data classes), a *Teleconference* component and forms to present Administrators and Units interfaces. The structure of the Control Centre is shown in figures 5 and 6.

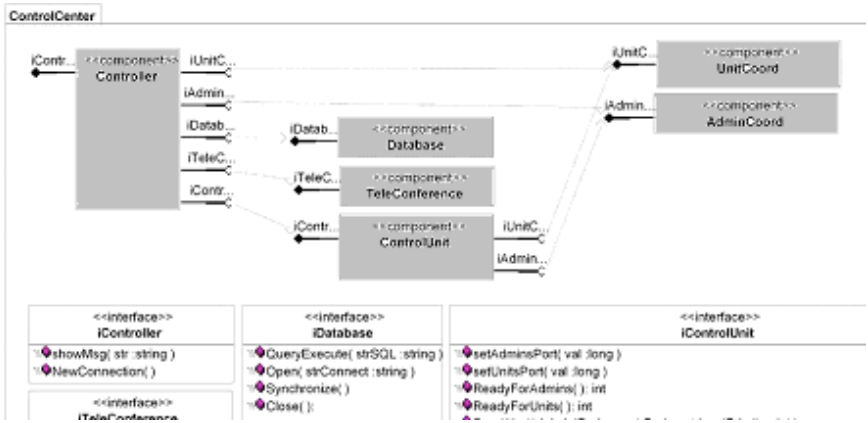


**Fig. 5.** Control Center Structure

The major functionality of the Control Centre is contained in the communicator switches and the controlling class. The controlling class receives a defined set of messages with which it updates local copies and forwards messages to Remote Units and Administrators. In our framework, all updating scenarios between the Tele-Medicine units that provide a consistent state of synchronized copies of Remote Units data to the administrators have been implemented. In these units the communication aspect is handled by the Control Unit and a Teleconference component that supports the media communication.

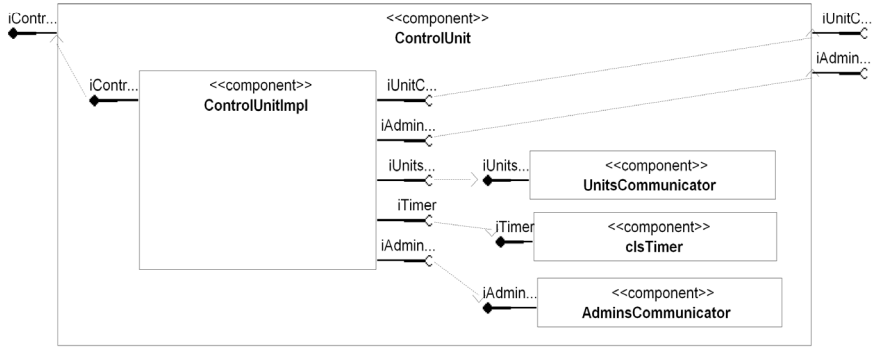


**Fig. 6.** Control Center – Control Unit

**Administrative Units.** The Administrative Unit structure is quite similar to the Remote Unit's one. The most significant role of these units is the fact that they are using a local copy of a Remote Unit's data. This copy is updated and informed by messages sent initially to the Control Centre and subsequently to the original Remote Unit. The Administrative Unit is using a Communicator to connect to the Control Centre, as in the case of the Remote Unit. The controlling form exists in the structure of this unit in order to be able to perform changes on copies of Remote Units. The

administrator is using two data classes that contains local information and the the copy of the selected Remote Unit that is currently handled by the Administrator. The structure of the Administrative Unit is shown in figures 7 and 8. Similarly to the Remote Units, a Control Unit component is used to handle communication and messaging, and a Teleconference component to support the teleconference communication with the Remote Units.

**Mobile Administrator.** The structure of Mobile Administrators is quite similar to the Administrative Units. They are using the same techniques to connect to the Control Center and exchange messages.

In the following sections, we will focus on the QoS issues for the Tele-Medicine framework and by using the above analysis, we will show how we proceed to derive the final QoS-aware framework.

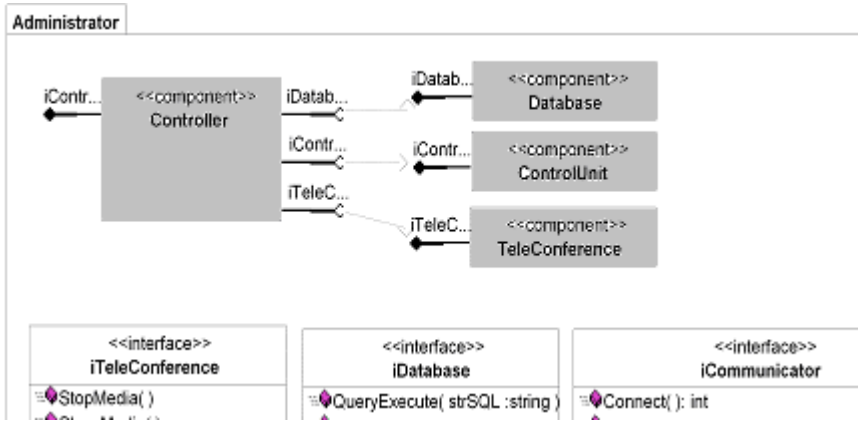


**Fig. 7.** Administrative Unit Structure

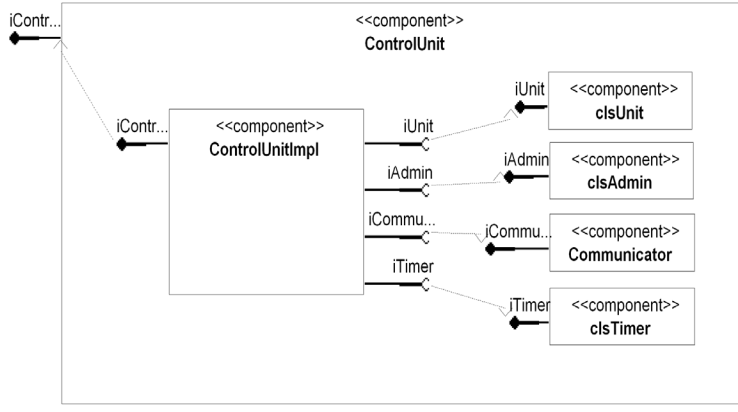


**Fig. 8.** Administrative Unit – Control Center

### 3.2 The QoS Requirements

The Tele-Medicine framework was fully operational but without any QoS guarantees. This resulted in an irritating behavior in the teleconferencing module during high network usage hours. The video was transmitted with delays and very low-resolution. Furthermore, again due to problems with the underlying network, the messages could take variable amount of time to arrive, disallowing the assumption that if a patient omits to take his medication or wants to send an urgent message to the doctor, the doctor will receive the message in a reasonably short amount of time.

The initially proposed solution for video-conferencing was to drop the resolution of the transmitted signal (video) or even drop video altogether, and transmit only sound. However, this would require a pre-test before starting the teleconferencing in order to estimate the speed of the network. But even in that case, the highly variable network quality could not be taken under consideration, that is, if during the pre-test the network was in bad shape, the transmission would be initiated with low-resolution video for the whole conferencing, and any improvement of the network status in the process would not be taken into consideration in order to switch to high-resolution. For the messages on the other hand, an easy way to guarantee a bounded maximum delay in transmission was by initiating a timeout thread that would raise an event after the timeout had expired, and force checking if the message was successfully sent. Both these solutions however are not optimal in matters of modularity and usability.

Furthermore, Tele-Medicine is a time-critical application. This means that information that arrives late and outside a specific time interval is not accurate and most probably not correct. The need for quality was obvious and the need for improving performance and stability begged for a better solution. Aspect Oriented Programming seemed to follow the requirements of such an application. However, the application was component-based and distributed, so we needed a methodology to consider the improvement of quality within the components itself ([10]). An AOP-based methodology was the suitable solution for this type of applications.

For these reasons, we used the QCCS methodology to improve the QoS aspect of the application. To solve the above mentioned problems, we implemented two contracts: (a) the network **Latency** contract, and (b) the **Bandwidth** contract. The latency contract is monitoring the network status periodically (not on a per message basis). Initiating from the time the connection is established, the network latency contract monitors the network status (through traditional SNMP ping messages) and reports to the user if the network bandwidth is lower than a threshold defined by the user. The above configuration is not computationally expensive and does not demand heavy network usage for the implementation. The bandwidth contract is also relevant to the network infrastructure. This contract is initiated when establishing a teleconferencing session. The contract is again periodically monitoring the available bandwidth of the two components that are teleconferencing (not the complete bandwidth but only the bandwidth available to the encoder and decoder components for video, e.g. downloading a file from the same machine at the same time would reduce this bandwidth). The resulting system is now able to adjust to different network conditions, dropping or increasing the resolution of the transmitted image. Furthermore, in case of failure in the latency contract (that is, the maximum time for successfully sending a message) the user is notified for further actions.

After deciding which contracts were required and suitable to be implemented, we needed to design new diagrams by using the QCCS methodology. First, we designed the new contracts (PIM) and then proceeded into transforming the PIM model into a PSM model.

# 4 Adding Quality Contracts

## 4.1 Designing the PIM

The PIM has been designed using an extended set of UML notations, the latter supported by an associated development tool which has been developed as part of the QCCS methodology. The QCCS developing environment supports contracts, contract types, and defining relations between components and contracts. Although modeling the PIM was rather straightforward because the Tele-Medicine framework was developed using OOP principles, it was not so clear how the new non-functional requirements (contracts) were expected to be applied in the improved QoS Tele-Medicine framework. The PIM model was an abstract design of the expected system that was also emphasizing the general issues of the architectural schema of the Tele-Medicine environment. The created PIM model proved to be very useful to the new application in that it helped us to understand and define the new enhanced features that were meant to be implemented as QoS contracts. The abstract analysis of the PIM models was an outline map that was excluding complex concepts of the framework.

The Remote Unit is containing a set of components to support the patients' home-based monitoring devices (Machine), the communication with the Control Centre, and a teleconference communication. Figures 3 and 4 show how these components use a set of interfaces to interact in order to create, parse and transmit signals and data through the contained Communicator component. We recall that in our system a dedicated component, Control Unit, is used to support communication with the Control Centre. This type of component exists in all types of Units but with a different interface. The role of these components is quite similar and the importance of their proper functioning crucial to the system. Figure 4 presents the internal structure of the Remote Unit and especially its Control Unit that handles a Communicator component to support the communication channel. In the following section, we will define how this type of component was used in order to support the QoS in latency violations. The teleconference component which was extensively used in the contracts is also further analyzed below.

So, Figures 5 and 6 show how these components use a set of interfaces to interact in order to create, parse and transmit signals and data between certain pairs of Remote Units and Administrators. In this specific case, we are using a Coordinator object to support the multiple connected units to the Control Centre. Here we employ multithreaded TCP client components responsible for supporting TCP communication instead of single Communicator components. Figure 6 shows the internal structure of the Control Center's Control Unit. This component is once again handling the communication with the connected Units.

The Administrative Unit structure is quite similar to the Remote Unit one. This unit includes a set of administration features to alter remote data but most of the internal classes are supporting the same role as those for the Remote Unit. Figures 7 and 10 show the structure of this unit and its Control Unit that parses and handles the communication messages.

### 4.2 Locating the Contracts

To define the contracts, we first identify what extra-functional requirements could help the project's functionality and value. During this step, the following requirements worth having were singled out:

- Timeout requirements: A process is given a certain amount of time to complete its work. The same applies to message transmission.
- Network quality requirements: Problems with network connections and failures are detected and either corrected or the user is notified about them.
- Multimedia requirements: The quality of the transmitted video is over a threshold.
- Availability requirements: Each component should be available for a minimum percentage of the overall time the application is running.
- Throughput requirements: Each component should satisfy a throughput for all the incoming requests.

In this paper, we will analyze two of the most important requirements and model them as contracts. As already mentioned, we selected the following two:

- Network quality requirements: We check the network status with simple SNMP (Simple Network Management Protocol) messages. By using ping messages, we are able to find the maximum latency between two points.
- Multimedia requirements: We periodically check the bandwidth between the two interacting points, the video encoder and the video decoder, and trigger an alternate behavior when the bandwidth was less than a threshold.

We are able to model these requirements as contracts, and add them to our PIM diagrams. The contracts were modeled in UML using stereotypes and added to the PIM. Obviously, the network contract could affect most of the components used for communication purposes and the multimedia contract would affect the specific communication components, which were used for teleconferencing. Below we present the new diagrams as they were updated to include the contracts. Figure 9 shows how the latency and bandwidth contracts are used in the Remote Unit's component structure. Here it is expected to have a set of handlers that can be triggered each time a bandwidth or latency violation occurs.

The contacts, as they were declared in the methodology, are meta-model components that are connected on selected interfaces. In our case, these contracts are linked to the Control Unit (latency contract) and on the Teleconference component (bandwidth contract). These contracts, as shown below, are linked onto those specific interfaces. It is like using a certain "minimal" set of interface properties and functions that are combined with the new PSM model that will be generated during the Model Transformation.

### 4.3 Modeling Contracts as Aspects and Generating the PSM

The contracts that were selected and modeled in the new PIM diagrams are the latency contract (for networking QoS) and the bandwidth contract (for multimedia QoS). These contracts were extensively analyzed in order to define not only how they will be embedded into the system but also which technologies will be used to support them. Examining these new features helped us to define the appropriate aspects that would support the model transformation.

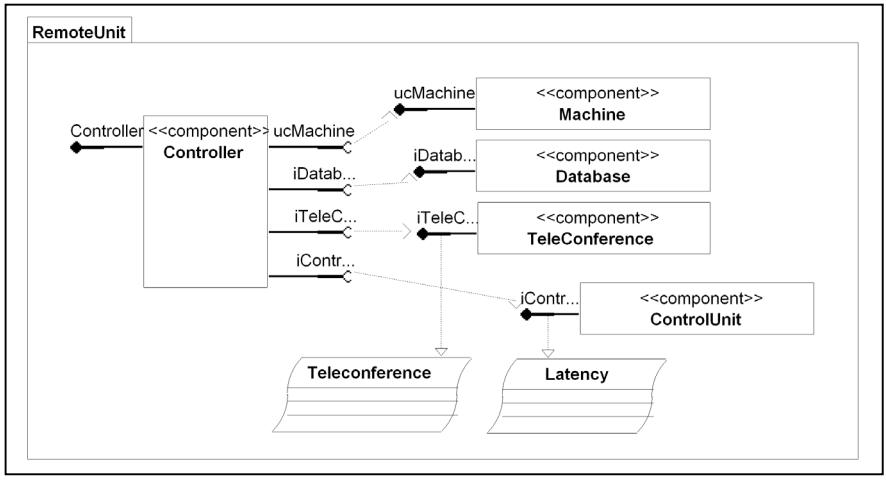


**Fig. 9.** Remote Unit with the Contracts

The contract functionality was designed and implemented using traditional OOP techniques. The contract code was implemented in C#, since we needed the generated code to be in C# too. We also identified how our existing components will be "communicating" with these new features of the system. This information was used to implement the aspect weavers (in Python) that were in turn used in the model transformation. This analysis helped us to design by "backward" analysis the aspects, but we succeeded in implementing abstract contracts that are reusable and efficient in similar applications that require pre-conditional interfaces. Figure 10 presents a first phase of the Latency Model Transformation. In the transformation, we use "replacements" that at the final stage will be substituted with classes and components. The aspect weavers are the transformation rules that will enhance the PIM model into a new PSM model that has predefined classes that occur due to the replacement of the contracts. The latency contract will produce a new class that is using a ConnectHandler and a DisconnectHandler to access the linked interface and embedded weaved code into the existing one. Therefore, the Control Units that were implemented support this required interface included in the latency contract, which is the pre-conditional requirement of the linked interface. The transformation that followed created the required code to support the QoS that was requested by the contract. Figure 10 consists of three parts. The first two represent the affected components of the PIM model; the third one is the "replacement" and "enhancement"

of the new PSM model that will be generated from this transformation. In the final PSM model a new component, the Latency Monitor, is embedded and "linked" to the connected interface in order to "trigger" a synchronous check of the communication channels. The Control Unit is enhanced with a link to a Latency Monitor that is activated and stopped by means of certain events. The most important element in this new connection is the Violation Handler that is "triggered" from the monitor automatically when the latency contact is violated. In Figure 11, the bandwidth contract is using a Connected and a Disconnected Handler to weave the required code into the existing one. In the final PSM model, similar to the previous analysis of the latency contract, a new component, the Bandwidth Monitor, is embedded and "linked" into the teleconference interface in order to "trigger" a synchronous check of the teleconference bandwidth. When the bandwidth is violated, a Bandwidth Violated event is generated that notifies the Teleconference component so that the latter can react appropriately.

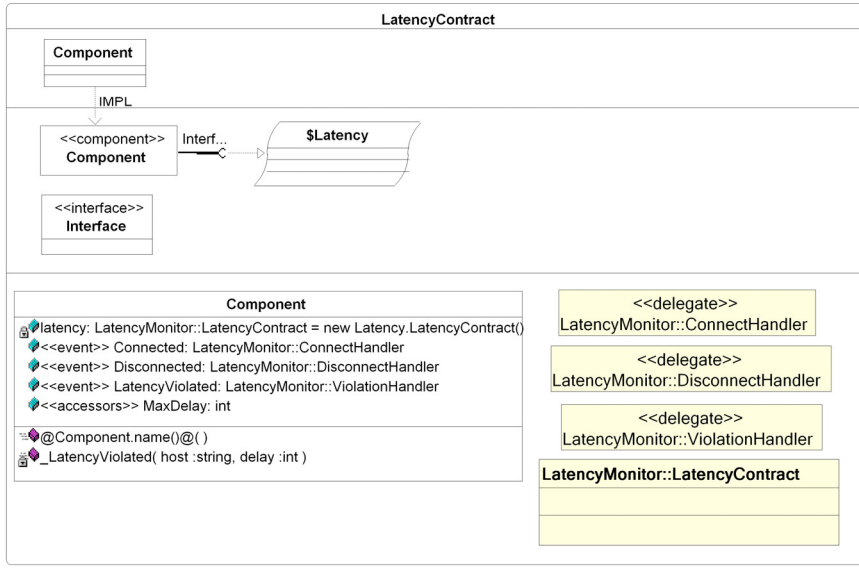


**Fig. 10.** The Latency Contract

The previous two transformations are composing a model transformer that was implemented in order to transform the PIM model into the new PSM model. This model transformer generates the new PSM model according to how it is expected to communicate with the other components of the system. The interfaces, ports and methods that were defined in the aspect analysis, were included into this new model to support the requested contracts and their role in the system. The previous transformation steps are the major steps that helped us to generate the final PSM model.

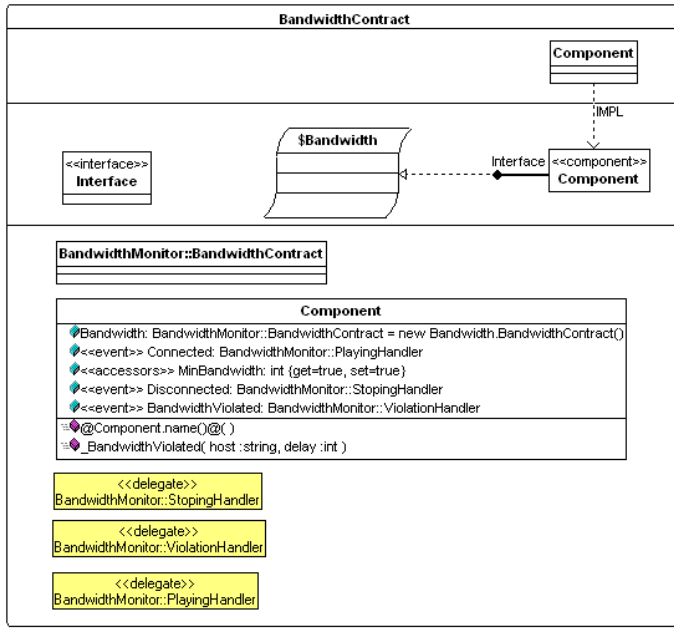


**Fig. 11.** Bandwidth Contract

### 4.4 Code Generation

The PSM model that was generated by the PIM transformation was not completely functional due to the complexity of our system. Code generation resulted in a skeleton schema to help us understand the system architecture that was expected to support the contracts. The existing code of the (non aspectual original version of the) implementation was easily imported into this structure and only the code handling "triggering" from the contracts was expected to be produced. These contracts produced the monitoring classes that were built to support synchronous checking on the required aspects (latency, bandwidth). At the end, the new "embedded" interface of the new linked monitor classes produced the proper triggers to produce the requested checking. We succeeded in enhancing our system by providing the following interesting features:

- Closing the communication channel when long delays occur. This is done automatically when the "generated" triggers discover that the SMTP ping communication has delays.
- In the teleconference communication, we used the trigger to adjust the media format (size, resolution) to the proper one by exchanging and changing the contract parameters. New messages were implemented to achieve an outstanding media "agreement" communication due to the available bandwidth.

It is worth pointing out how easy it was to use the enhanced coding that was generated from the aspect weavers. The difficult part is the implementation of such transformations. We must be strict in the pre-conditional linked interface structures but these transformers can easily enhance any similar application that requires these QoS contracts.

## 5 Conclusions and Further Work

The proposed methodology was found to be very effective for the implementation of quality-aware components. More specifically, the two contracts that were modeled in the design phase were successfully integrated into the final system, making the related components quality-aware. We would like to briefly report on the following subjects: (a) ease of distinguishing the contracts, (b) ease of developing the contracts, (c) ease of using the contracts, (d) size of code needed to develop for quality-awareness, and (e) reusability of the contracts. Using the contracts to derive quality-aware components was easy. The whole system was modeled in UML (platform independent models) and the model was later enhanced with the contracts. Then, we were able to extend our initial PIM to a PSM (platform specific model) with code and other platform specific information, and finally generate the complete system architecture. The code needed to be manually written for the implementation of the quality-aware components was not of significant size. This included the code that was used for the construction of the contracts and the code that was manually written in order to use the contracts in the final application.

Finally, the contracts have a high degree of reusability, since they were not developed for the selected application only. For instance, the contract that was monitoring the network latency is easily reusable in any other .NET application. To conclude with, we found this methodology not only efficient for quality-aware software, but also easy to learn, use, and incorporate into the business model. The development of a "contract" library, ready-to-use with a variety of applications, could help the component software industry in general. Therefore, we strongly believe that this QoS analysis and implementation technique is a viable methodology, and, being aware of current difficulties in the field, we expect to see it used in the analysis field soon.

The issue of QoS and its relationship with AOP is also studied in [13]. The work describes an adaptive environment, where QoS management is modelled as an aspect. The code of the latter can then access application-specific middleware and system information and controls, without the need to modify the actual application code. The associated aspect model includes join points specific to distribution and adaptation. Another approach on dealing with QoS-related non-functional aspects is introducing the notion of a component *container* ([14]). The different aspects are woven into the running system by means of such a container which provides concepts for negotiation and monitoring of non-functional contracts between different components and operating system resources, as well as other platform services. This work focuses in particular on the streaming interfaces of multimedia-based applications.

The implementation of QoS components based on contracts is still improving and the associated software analysis and design techniques have not yet been standardized. There is a lot of research to be done, to provide not only standards in analyzing such QoS issues but also a "ready-made" set of contracts that will include generators and transformers that will initiate "active" monitors for QoS aspects. Initially these contracts definitions should be independent of any programming language, in order to provide the required interface of designing a proper PIM model. Only when the developer decides which features should be supported by the implementation platform, then these features should become part of the PSM. We

believe that using such techniques will enhance the use of QoS aspects and will lead to the development of more reliable, consistent and high performance applications.

## References


1. L. Bass, P. Clements and R. Kazman: Software Architecture in Practice, Addison-Wesley (1998).
2. F. Halsall: Data Communications, Computer Networks and Open Systems, Addison-Wesley (1996).
3. E. Bneton and M. Rveill: An Architecture for Extensible Middleware Platforms, *Software: Practice and Experience*, Vol. 31 (13), 2001, pp. 1237-1264.
4. W. Lowe, M. L. Noga, A Lightweight XML-based Middleware Architecture, 20th IASTED International Multi-Conference Applied Informatics (2002), ACTA Press.
5. G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J-M Loingtier, and J. Irwin. Aspect-Oriented Programming. ECOOP '97, Jyväskylä, Finland, June, 1997, LNCS 1241, Springer Verlag, pp. 220-242.
6. A. Le Guennec, G. Sunyé, and J-M. Jézéquel, Precise Modeling of Design Patterns, UML 2000, LNCS 1939, Springer Verlag, pp. 482-496.
7. P. Fradet and M. Südholt. AOP: Towards a Generic Framework Using Program Transformation and Analysis. In ECOOP'98 Workshop on Aspect-Oriented Programming, 1998.
8. T. Weis, C. Becker, K. Geihs and N. Plouzeau, An UML Metamodel for Contract Aware Components, UML'2001, Canada, LNCS 2185, Springer Verlag, pp. 442-456.
9. B. Meyer, Applying Design by Contract. IEEE Computer Special Issue on Inheritance and Classification, 1992. 25(10), pp. 40-52.
10. S. Frolund and J. Koistinen, Quality of Service Specification in Distributed Object Systems. *Distributed Systems Engineering*, Vol. 5(4), 1998, pp. 179-202.
11. OMG. UML notation guide.
12. QCCS, http://www.qccs.org.
13. G Duzan, J. Loyall, R. Schantz, R. Shapiro and J. Zinky, Building Adaptive Distributed Applications with Middleware and Aspects, AOSD 2004, March, 2004, Lancaster UK, ACM Press, pp. 66-73.
14. S. Gobel, C. Pohl, S. Rottger and S. Zschaler, The COMQUAD Component Model, AOSD 2004, March, 2004, Lancaster UK, ACM Press, pp. 74-82.