QoS-aware Mobile Middleware for Video Streaming

Sten L. Amundsen, Ketil Lund
Simula Research Laboratory
{stena, ketillu}@simula.no

Carsten Griwodz, Pål Halvorsen
Department of Informatics, University of Oslo
{griff, paalh}@ifi.uio.no

Abstract

State-of-the-art middleware and component technologies lack support for Quality of Service (QoS) management. Application developers, therefore, integrate QoS mechanisms into the application itself. In this paper, we propose a solution for how a mobile middleware can take on the responsibility for QoS management. We use a video streaming scenario to identify the QoS mechanisms that the middleware must manage on behalf of the application, and we demonstrate the feasibility of our solution within this scenario. The key concept of our work is the service plan that specify the service and the QoS of components and compositions. Using service plans recursively, we model alternative application configurations such that a QoS-aware middleware platform can safely configure and dynamically reconfigure applications, based on user requirements and resource availability. In this paper, we show that service plans enables the separation of application design from QoS management, in a way that promote reuse of both functional code and QoS mechanisms.

1 Introduction

Many new applications meet stringent requirements to be able to provide services that users perceive as good quality. To meet the increasing performance and scalability requirements, mechanisms that adapt to various workloads and requirements must be designed. One must also develop strategies for dynamically combining components into a high-performance service suitable for the underlying infrastructure.

Because most existing infrastructures offer no quality of service (QoS) guarantees, adaptation mechanisms are needed to maintain QoS. The traditional way of handling this is to integrate QoS mechanisms with the application logic, i.e., making the components self-adaptive. This gives customised QoS management mechanisms for the application in question. It also makes the application complex and hard to manage, and the implementation of QoS mechanisms cannot be reused in other applications. Our approach is to separate the application logic from the domain specific QoS management, and instead deploy alternative application configurations with different QoS characteristics on a QoS-aware middleware. By combining this with rigorous service- and QoS-modelling we make it easier to reuse both application components and QoS mechanisms, and ensure safe reconfiguration at runtime.

In this article, we employ a scenario to identify the QoS mechanisms that the middleware must manage on behalf of the application and the appropriate method for service and QoS modelling. We choose to combine the streaming application domain with the mobile technical domain, since this gives a scenario where QoS mechanisms and dynamic reconfiguration are particularly useful.

The article is structured as follows. Section 2 outlines the scenario, and section 3 describes our proposed solution for achieving QoS in the scenario. In section 4, we describe the realization of the middleware and application, including service and QoS modelling. Section 5 discusses related work, and lastly, section 6 gives some conclusions and the direction for further research.

2 Scenario Description

Recent advances in wireless networking technologies have enabled the deployment of video streaming applications in the mobile domain, which raises several new challenges in order to achieve best possible playback at the terminal. To make these challenges explicit we use a scenario. In this section we present an overview of the scenario, and for a detailed description, we refer to our technical report [1].

2.1 Video Streaming to Mobile Terminals

Clients access the video server from different terminals types: home theaters, laptops, and personal digital assistants (PDAs), which are connected to the Internet over the access networks: fixed local area network (LAN), wireless LAN (WLAN), and general packet radio service (GPRS) in GSM (see Figure 1). IP mobility management enables

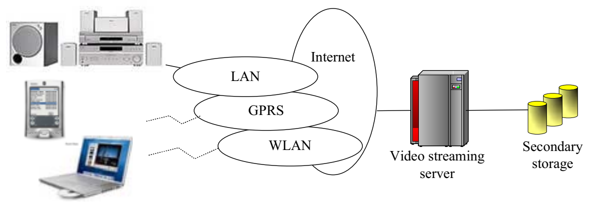

Figure 1. System overview

the mobile terminals, PDA and laptop, to roam seamlessly between the access networks [2]. The main challenge is to give the users a high-quality playback in different contexts: home theater-LAN, laptop-LAN, laptop-WLAN, PDA-WLAN, PDA-GPRS, when network conditions are changing and users roam between access networks.

2.2 Dynamics and Constraints

In the video streaming scenario, there are several challenges. One is the traditional streaming requirement, timely delivery of high data rate streams. Another, more complicated, is the handling of the different combinations of access networks and terminal types. Assume for example that we have a video server capable of delivering MPEG-II DVD-quality movies. In addition to requiring substantial memory and processing resources, it must be capable of transmitting streams at a rate of 4-8 Mbit/s on average and a maximum of 11.8 Mbit/s. Such streams might be requested by high-end systems connected to high bandwidth access networks, but in case of mobile terminals connected through a wireless network, neither the network nor the terminals have the required resources. Thus, trying to send a full quality stream over a dedicated GPRS link (of maximum 107.2 kbit/s, using eight 13.4 kbit/s time-slots) to a PDA (e.g., PalmOne Treo 650) is not feasible.

Each user has their own opinion of what high quality is, e.g., different values for QoS dimensions like frame rate, resolution and color depth. Thus, streaming the same content to users using the same technology may result in streams with different characteristics and requirements. Hence, the application must be adapted to the user's QoS requirements and the capabilities of all the resources along the data path from server to client. In our scenario, a terminal connected to a LAN needs a media player to playback the received video stream, while a mobile user with a PDA connected to a WLAN possibly needs a different codec, as well as forward error correction (FEC) due to a higher packet error rate and longer retransmission delay.

To manage QoS in these contexts, an understanding of the QoS characteristics is needed. The application and resource QoS characteristics are therefore analyzed (results are shown in Figure 2. We adhere to OMG's UML profile

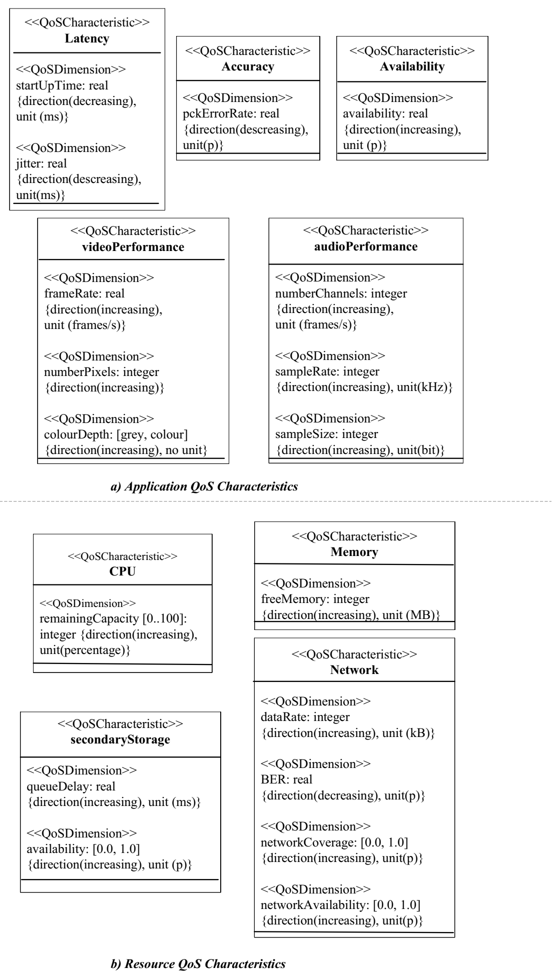
<<QoSCharacteristic>> Latency
<<QoSDimension>> startUpTime: real {direction(decreasing), unit (ms)}
<<QoSDimension>> jitter: real {direction(descreasing), unit(ms)}
<<QoSCharacteristic>> Accuracy
<<QoSDimension>> pckErrorRate: real {direction(descreasing), unit(p)}
<<QoSCharacteristic>> Availability
<<QoSDimension>> availability: real {direction(increasing), unit (p)}
<<QoSCharacteristic>> videoPerformance
<<QoSDimension>> frameRate: real {direction(increasing), unit (frames/s)}
<<QoSDimension>> numberPixels: integer {direction(increasing)}
<<QoSDimension>> colourDepth: [grey, colour] {direction(increasing), no unit}
<<QoSCharacteristic>> audioPerformance
<<QoSDimension>> numberChannels: integer {direction(increasing), unit (frames/s)}
<<QoSDimension>> sampleRate: integer {direction(increasing), unit(kHz)}
<<QoSDimension>> sampleSize: integer {direction(increasing), unit(bit)}
a) Application QoS Characteristics
<<QoSCharacteristic>> CPU
<<QoSDimension>> remainingCapacity [0..100]: integer {direction(increasing), unit(percentage)}
<<QoSCharacteristic>> Memory
<<QoSDimension>> freeMemory: integer {direction(increasing), unit (MB)}
<<QoSCharacteristic>> Network
<<QoSDimension>> dataRate: integer {direction(increasing), unit (kB)}
<<QoSDimension>> BER: real {direction(decreasing), unit(p)}
<<QoSDimension>> networkCoverage: [0.0, 1.0] {direction(increasing), unit(p)}
<<QoSDimension>> networkAvailability: [0.0, 1.0] {direction(increasing), unit(p)}
<<QoSCharacteristic>> secondaryStorage
<<QoSDimension>> queueDelay: real {direction(increasing), unit (ms)}
<<QoSDimension>> availability: [0.0, 1.0] {direction(increasing), unit (p)}
b) Resource QoS Characteristics

Figure 2. QoS characteristics

for QoS modeling [3], because i) it is a formal specification that defines the terms and meta-models we need and ii) this ensures that our QoS models can be integrated into design models and existing software development methods.

3 Proposed Solution

We advocate that the *middleware* should select and combine the most appropriate components for a given context (e.g, access network technology, execution environment, and terminal type) and available resource capacity (e.g., CPU, storage, and network).

If a terminal stays connected to the same network, the initial configuration will remain fixed during the whole session. However, with mobility follows a much more dynamic environment, and the middleware must be able to adapt the application to context changes and resource fluctuations, in order to maintain the best possible QoS. In general, there are two adaptation types that must be supported:

- changing parameter settings of individual components
- changing the application composition

Each application variant, resulting from performing one of these types of adaptation, is called an *application configuration*. We also recognize that some QoS mechanisms are so tightly coupled with the corresponding functionality that separating them out would result in a very complex design. One example of such a mechanism is data traffic control, which constantly monitors round trip time and packet error rates, to adjust the transmission rate. Consequently, our approach to handle the dynamics in the scenario is to let the middleware control QoS to the extent feasible, and to allow component self-adaptation where necessary. This requires that the QoS characteristics is specified for each component, which then are used by the middleware to assess the suitability of each component when assembling a composition. For self-adapting components, it is important that the ability to self-adapt is expressed and quantified in the QoS characteristics.

In many cases, application developers want to reuse existing code (e.g., video codecs). Hence, existing code is first wrapped to make it a first-class component, before a QoS expert analyzes the component and then attaches a service plan that describes the QoS characteristics of the component.

Finally, it is crucial that the video streaming application is stable, during and after a reconfiguration. To achieve this, alternative configurations are rigorously modelled, which ensures that only component compositions and configuration values defined at design time are used. We call this property *safe reconfiguration*, and together with platform-managed adaptation it represents the contributions of our work.

4 Realization

4.1 QoS-aware Middleware

We have designed and implemented a new component-based QoS-aware middleware, based on an open, reflective component architecture, called QuA (Quality of service-aware component Architecture), with hooks where QoS management components can be inserted as plug-ins [4, 5], as shown in Figure 3. To make the middleware executable on both mobile terminals and large servers, the architecture has a small core, and everything else is provided as pluggable components. The middleware has been implemented in both Java and Smalltalk, and we have published the Smalltalk source code together with the platform independent model (PIM) [6].

QuA makes QoS decisions that take advantage of runtime information, about context and resources, to select

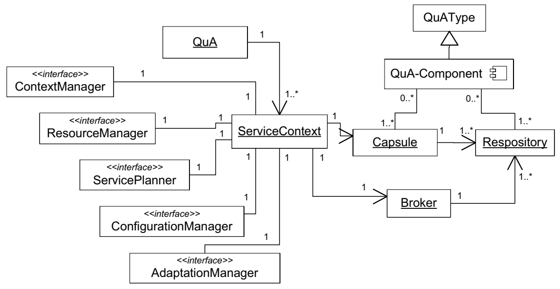

Figure 3. Middleware overview

the application configuration suitable for the current context and resource availability and that meet the user's QoS requirements. Fundamental in this approach is to model the application as service types and at runtime let the middleware select among alternative implementations of these types. The alternative implementations, each with different QoS characteristics, are provided by application developers and deployed in a repository, together with associated *service plans*. Service plans play a central role in QuA, and they serve three purposes: i) provide the link between a service type and an implementation of the type; ii) specify service composition and parameter configuration of the implementation; and iii) describe the QoS characteristics of the implementation.

As shown in Figure 4, the service plan contains eight information elements: i) *dependencies*: requirements to the execution environment, libraries, and static dependencies to front-end or back-end systems; ii) *parameter configuration*: parameters the component composition or component is to be configured with; iii) *composition specification*: a graph specifying the construction of the service, i.e., the composition of service types and the bindings between them; iv) *role*: a role name space and role names for service types and component types in the composition. The same role name in two alternative service plans will during reconfiguration be interpreted as identical services and, hence, not be replaced during dynamic reconfiguration; v) *offered services*: services/operations that the composition/component offers; vi) *input QoS contract*: QoS values along QoS dimensions that users of this composition/component must adhere to; vii) *QoS model*: a model of the QoS characteristics, which is defined using QoS dimensions independently of the execution environment. The model specifies the possible range of QoS values along the QoS dimensions; and viii) *QoS mapping*: functions that establish the logical relationships between QoS characteristics at different levels.

Users specify their QoS requirements in a user QoS specification using *dimensional utility functions*, which give users the means to specify their preferences at a high abstraction level. In addition to specify the QoS requirements along the user QoS dimensions, these functions capture the user's trade-off between the dimensions. This enables QuA

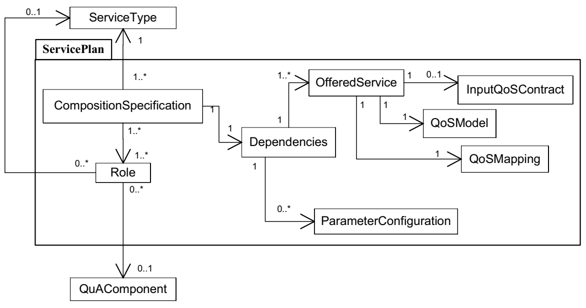

Figure 4. Service plan

to find the resource allocation that both meet the user's requirements and optimize the resource utilization with respect to the user experience.

When a user request a service, QuA invokes the *service planner* with the user's service request (i.e., the service type and QoS specification). The planner asks the *broker* to provide plans that implement the requested type, and the broker searches the *repository* for relevant plans. The results are one or more plans being returned to the service planner, each with different resource requirements and QoS characteristics. It should be noted that most services are compositions components, the returned plans normally are recursive that the service planner combines into complete service plans. The service planner now uses the utility function from the user and the QoS mapping functions to assess the suitability of each complete plan. From the *context manager* plug-in it receives information about the hardware and execution environment, while the *resource manager* plug-in provides information about availability of the different resources. Thus, the service planner is able to select the service plan that provides best utility for the user in the current context and resource situation [7].

Next, the *configuration manager* uses the selected complete service plan to instantiate and configure the requested service. If the service is a composition, the configuration manager starts with the plans for the atomic services, and build the composition bottom-up.

Finally, to enable dynamic reconfiguration, sensing agents and resource monitors are deployed as component types in the repository. This enables QuA to discover context changes and resource fluctuations, and if necessary to maintain the QoS-level re-plan the service. Dynamic reconfiguration, i.e., either by component parameters changes or application re-composition, is managed by the *adaptation manager* (see Sub-section 4.3).

4.2 Application Design

The application must be designed with reconfiguration in mind, and herein lays the crux of our solution for QoS-awareness and safe reconfiguration. The design phase has five steps: i) defining components, ii) specifying compositions, iii) specifying parameter configurations, iv) service modelling, and v) QoS modelling. Each step is described in the following paragraphs, starting with existing code from other streaming applications that are reused and encapsulated in QuA components.

Defining components. For signalling we use an existing implementation of the real-time streaming protocol (RTSP), the RSTP client and server code from *komssys* [8]. Encoded video is transported over the real-time transport protocol (RTP) extended with a TCP friendly flow control algorithm [9]. The flow control adjusts the transmission rate to the network data rate by measuring round trip times and packet loss. To choose the appropriate video format and decoder we conducted a series of tests [1], and found that *mencoder* to pre-code the video files in MPEG-4 format and *mplayer* [10] as sink component were suitable for our scenario. To protect the video from packet errors we apply a FEC algorithm, Reed-Solomon erasure correction [11]. Pre-coded videos stored in the secondary storage are accessed by components executing on the QuA middleware. This enables us to deploy alternative versions (resolution, rate and colour) for a movie title and during runtime choose one video version based on the resulting QoS.

Specifying compositions. From the components, atomic services are defined, which are combined into three alternative service compositions, illustrated in Figure 5. Each composition has distinct application QoS characteristics and resource requirements, which corresponds to the resource QoS characteristics of the context[1].

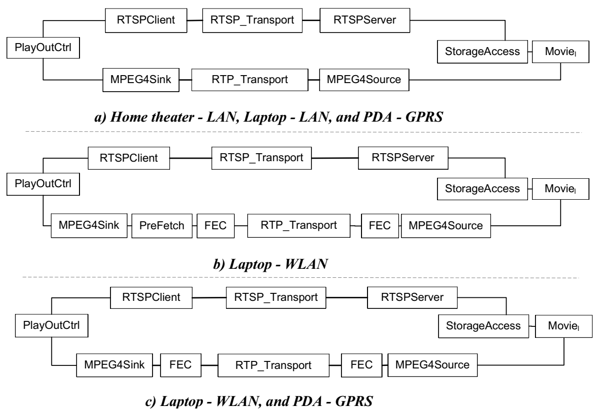

Figure 5. Component compositions

[1]For example, the PDA & GPRS context can only sustain the memory requirements in compositions a) and c) in Figure 5, since pre-fetching is too memory demanding for a PDA. In the system context PDA & GPRS, the network data rate varies to such an extent that two alternative configurations are required; one for data rate of 42-56 kbit/s (2.5G), and the other for 100-384 kbit/s (3G).

Specifying parameter configurations. The components that implement the atomic services *MPEG4Source, MPEG4Sink* and *FEC*, have alternative parameter configurations. The source and sink components are configured with resolution, colour depth, and frame rate, while the *FEC* component is configured with the number of data- and parity packets and the packet size.

Service modelling. Atomic services are grouped into sub-services that again are grouped to the service offered to the user. For signalling, there is only one possible composition, hence, the sub-service is associated with a fixed set of atomic services. For streaming and movie, alternative compositions and parameter configurations are identified. These alternatives are specified in different service plans, which are deployed in the QuA repository with an association to the service type. Figure 6 illustrates how one of the alternative service plans, *MPEG4Simple*, is associated with service types at the atomic and sub-service levels. Similarly, for the movie, there are alternative implementations of the sub-service, $Movie$, and the atomic service, $Movie_l$, as shown in Figure 7. All sub-service types are then combined to construct the video streaming application, as illustrated in Figure 8.

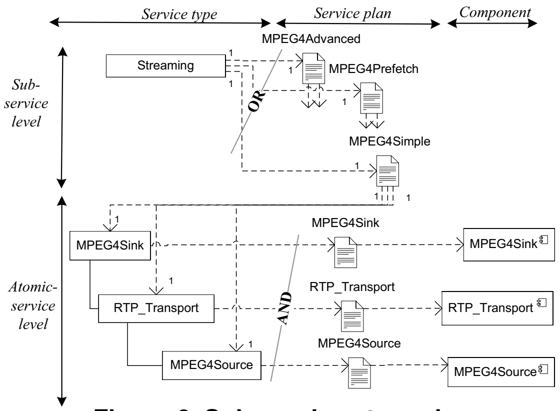

Figure 6. Sub-service streaming

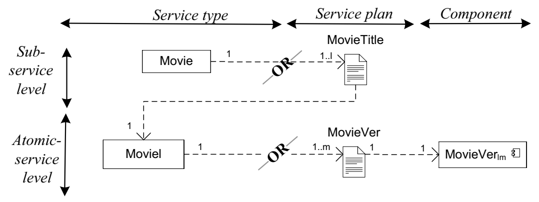

Figure 7. Sub-service movie

QoS modelling. The QoS characteristics at each service level are modelled: service, sub-service and atomic service, plus the parameter configurations to the components. To illustrate how the QoS characteristics are identified and specified, we describe the QoS modelling of the *MPEG4Sink* component and the *MPEG4Simple* implementation of the streaming sub-service. For the complete QoS modelling, see [1].

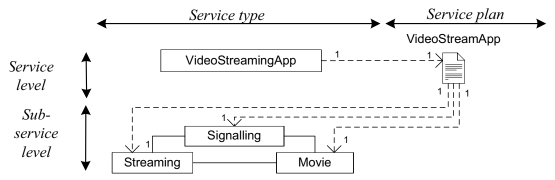

Figure 8. Service and Sub-service level

The *MPEG4Sink* component offers one service, the operation *forward()*. The component is configured to a combination of resolution, frame rate and colour depth that corresponds to a version of the movie title, $MovieVer_{lm}$. Resolution is the number of pixels on the screen, rate is the video playback speed in frames per second, and colour depth is either greyscale or colour. The QoS dimensions for the offered service, *forward()*, are identified by studying the relationship between input messages to and output messages from the component. For the *MPEG4Sink* component, only one QoS dimension is found from the difference between input and output messages, namely start-up time. The acceptable range of QoS values is set to [0, 2400] (lowest value gives highest QoS). For video decoding, it is not feasible to define general mapping functions, due to the discrete cosine transformation and the prediction frames. Hence, the resource requirements are identified in a series of tests. Table 1 lists measurements for the decoding of an episode from the science fiction series "Andromeda". The tests were performed on a standard PC with an Intel Pentium 4 2.4 GHz and Redhat Linux OS version 2.4.21-20.EL. These values are also used as rough indicators for other combinations of hardware and OS. CPU requirements are mean percentage allocation, memory requirements are peak usage, and network requirements, v_{bit_rate}, specify the mean throughput at the application level. These measurements are stored in the service plan, in the QoS mapping information element, together with a mapping function that relates QoS values to the measured resource requirements. Start-up time is mapped to the time used to decode video frames in the byte stream, which depends on the GOP/VOP pattern and the number of input messages needed before coding can start. We make a rough approximation, and assume five frames between each I-frame, i.e., five frames are clocked in before decoding starts. It is also assumed that the time used for decoding is negligible, i.e., do not consider internal processing, and instead predict the time needed to fill up the component with sufficient volume of encoded video.

The resulting QoS mapping function, which refers to the table with measured resource requirements, is shown in Equation (1). It uses the QoS mapping function to *RTP_Transport* to predict the throughput in the current context. We refer to [1] for a description of this QoS mapping

Movie version			Resource requirements		
Resolution (pixels)	Rate (fps)	Colour (colour/grey)	CPU (%)	MEM (kB)	Vbit_rate (kB/s)
1440x1152	25	colour	0,3625	34128	225
720x576	25	colour	0,0375	34128	224
"	"	grey	0,0307	34128	224
"	15	colour	0,0328	34128	223
"	"	grey	0,0276	34128	223
"	5	colour	0,0216	34128	113
352x288	25	colour	0,0040	34128	136
176x144	25	colour	0,0003	34128	53
100x100	20	colour	0,0003	34128	25
"	15	"	0,0003	34128	21
"	10	"	0,0002	34128	15
"	5	"	0,0002	34128	9

Table 1. Configurations and resource use

function.

$$\begin{aligned} t_{MPEG4sink\_start} &= \frac{5}{Rate}\frac{v_{bit\_rate}}{R_{RTP\_Transport}}(ms) \\ &\forall CPU \geq g(resolution, rate, colour), \\ &MEM \geq f(resolution, rate, colour), \\ &R_{RTP\_Transport} \leq v_{bit\_rate} \end{aligned} \tag{1}$$

The *MPEG4Sink* atomic service is used in the composition to the sub-service *MPEG4Simple*, together with *RTP_Transport* and *MPEG4Source*. The sub-service offers one service, *stream()* (operation on the *MPEG4Source* component). The input-output relationship for this service gives four QoS dimensions, one of which is start-up time. The QoS dimensions are mapped to the service plans that specify the implementation of the three atomic services, examplified for start-up time in Equation (2). The QoS model and QoS mapping functions for the *VideoStreamingApp* service are defined in a similar fashion.

$$\begin{aligned} t_{MPEG4S\_start} &= t_{MPEG4source\_start} + \\ &t_{RTP\_Transport} + t_{MPEG4sink\_start(ms)} \end{aligned} \tag{2}$$

4.3 Dynamic Reconfiguration

In the previous sub-section, we described how the application is designed and deployed as a set of alternative configurations. After instantiating the application, the information used during the instantiation process is retained by QuA (user QoS requirements, complete service plan, and references to all component instances). This information is meta-data, and is used for both types of reconfiguration distinguished in Section 3, in order to achieve an efficient transition between configurations. For changing component parameter settings, the composition specification (in the complete service plan) and references to component instances are used to locate the correct instance to reconfigure. When altering the composition of the application, it is a goal to minimize the number of modifications that must be made. For example, when going from a composition with *FEC* components to one without (see Figure 5c and 5a), it is preferable to retain all other components and just add new bindings in place of the *FEC* components. The means to achieve this is through the composition specifications of the two configurations. Since we require that alternative composition specifications belong to the same role name space, QuA knows which roles, and thereby component instances and bindings, that can be retained in the new configuration.

Dynamic reconfiguration starts when the adaptation manager is notified by the context manager or resource manager about changes that may affect the utility of the application. The adaptation manager then asks the service planner to re-evaluate the utility of the running configuration, using the original service request and QoS requirements as input. If the result indicates a sufficiently large drop in utility, the adaptation manager asks the service planner to re-plan the service. The new plan is forwarded to the configuration manager, which compares it to the running configuration, and determines which changes that has to be made. Through the safe reconfiguration property, we ensure that the new configuration does not leave the application unstable, and where required and possible, state is transferred from the old to the new application instance. Finally, the configuration manager makes the necessary changes to the running application, before the meta-data is updated to reflect the new configuration.

5 Related Work

Several research activities have addressed mechanisms for video streaming to mobile terminals, but for the work presented in this article only results within the area of dynamic streaming applications are relevant. InfoPipes [12] is one such result, since it employs a dynamic framework for composing a streaming service that processes the media stream as it flows through a pipeline of components. Another is [13], a scaleable and fault tolerant multimedia distribution system that add, remove, and replace stream handlers. QuA also chose and reconfigure service compositions, but the decisions are based on QoS information and not functional properties, taking video streaming one step further.

With respect to general component architectures, Open-ORB v2 addresses QoS management by introducing component frameworks (CF) as building blocks [14]. Each CF has a set of policies and rules that provides QoS-support. The QuA middleware support CFs, but the QoS mechanisms are separated from the application and into the middleware. The middleware platforms OpenORB [14], CARISMA [15], and DynamicTAO [16] employ reflection for dynamic reconfiguring. In OpenORB, applications can reconfigure the structure of the CFs, while CARISMA uses reflection to add or change policies in the QoS-profile associated with the application. DynamicTAO adds reflection to CORBA, allowing inspection and reconfiguration of the ORB. All these systems require application code for QoS handling. The QuA platform, on the other hand, does not require any QoS-related application code, since the service plans capture the QoS-characteristics of the applications.

Schantz et al. [17] describe patterns for reuse of QoS management mechanisms in the QuO CORBA middleware, an approach that is extended to the CORBA component model [18]. Necessary QoS management functionality is deployed inside the adaptive components, which is fundamentally different from QuA, where separation of concerns is achieved by offering platform-managed QoS to the application. Mitchell et al. [19] use a runtime model of component resource requirements (CPU) that, similar to the QuA service plan, is separated from the application logic. However, in QuA the service plan also encompasses QoS mapping and hierarchical construction of alternative subservices from atomic components.

Odyssey [20] uses fidelity, a data quality measure (video frame rate, image resolution, and voice quality) to control what data to retrieve and from where. When a client is accessing the remote server, Odyssey finds a pre-processed video file, suitable for downloading over the given connection, and downloads it. The principle of choosing versions of the video is also present in our work, but the difference is that Odyssey is tightly coupled to an open source OS and the adaptation is limited to content. Barwan [21] and Prayer [22] insert a proxy server in the data flow that performs video transcoding and protocol adaptation, which makes the video servers unaware of the wireless access network. This is a viable approach, though only useful for streaming.

In addition to dynamic middleware, frameworks are available for identifying and choosing service compositions at runtime. For example, SpiderNet [23], designed for overlay networks, combines 1) a layered system model that relates service composition (a graph specified by the user) and components, 2) a service component model that encapsulate both functionality and meta-data, and 3) a probe protocol that performs admission control and resource allocation along the route. QuA, designed for QoS-aware middleware, has a more rigorous way of defining alternative compositions and meta-data (service plan), which provides separation of concern and the safe reconfiguration property.

6 Conclusions and Future Work

In this paper, we have demonstrated that it is feasible to separate domain-specific QoS mechanisms from the application logic, even in a highly dynamic environment like the mobile domain. QuA, with its QoS management plugins, incorporates the QoS mechanisms that are traditionally entangled with the application logic, and manages QoS at runtime, on behalf of the application. To provide each user with best possible perceived video quality, each new instance of the streaming service is composed according to the preferences of the user, taking current context and available resources into consideration. QuA uses QoS mapping functions to translate the QoS dimensions specified in the user preferences into resource requirements, and select the application configuration that best meets the user's requirements. The scenario also describes and exemplifies activities and models that the application developer must take into consideration at design time, to ensure safe reconfiguration. Our conclusion from the scenario is that this approach is both feasible and advantageous.

General results are the principles for service modelling, QoS modelling, and the recursive service plans. Our service modelling allows application developers to design applications with alternative component compositions and parameter configurations. Modelling of the QoS characteristics of the different alternatives captures the QoS expert knowledge about the performance of the component, where the quality of the prediction made by QoS mapping functions is essential for the middleware to make sound QoS-related decisions. Deployable service plans contain the results from both service and QoS modelling, and make the logical connection between service types and their alternative implementations. Together, this ensures that a QoS-aware middleware, like QuA, can choose the composition and parameter configuration suitable for the current context and resource availability.

Currently, our research is addressing runtime considerations, with emphasis on extending the QuA implementation with support for re-planning and dynamic reconfiguration of parameter configurations and component compositions.

7 Acknowledgements

The QuA base-line is designed and written by Richard Staehli, who also helped us in understanding the importance of service plans and modelling of the non-functional characteristics. Thanks also to Prof. Frank Eliassen for making the cooperation between Simula and University of Oslo happen, and providing insightful comments to our work that substantially improved our solution and this paper.

References

[1] S. Amundsen, K. Lund, C. Griwodz, and P. Halvorsen. Scenario Description-Video Streaming in the Mobile Domain. http://www.simula.no:8888/QuA/2/techVScenA1.pdf, 2005.

[2] Charles Perkins. IP Mobility Support for IPv4. IETF RFC 3344, August 2002.

[3] OMG. UML Profile for Modelling Quality of Service and Fault Tolerance Characteristics and Mechanisms. OMG Adopted Specification, June 2004.

[4] S. Amundsen A. Solberg, J. Ø. Aagedal, and F. Eliassen. A Framework for QoS-Aware Service Composition. In *Proceedings of 2nd ACM International Conference on Service Oriented Computing*, 2004.

[5] R. Staehli, F. Eliassen, and S. Amundsen. Designing Adaptive Middleware for Reuse. In *Proceeding from 3rd International Workshop on Reflective and Adaptive Middleware*, 2004.

[6] Simula Research Laboratory. QuA documentation. http://www.simula.no:8888/QuA/55, 2004.

[7] S. Amundsen, K. Lund, F. Eliassen, and R. Staehli. QuA: Platform-Managed QoS for Component Architectures. In *Proceedings from Norwegian Informatics Conference (NIK)*, pages 55–66, November 2004.

[8] M. Zink, C. Griwodz, and R. Steinmetz. KOM Player -A Platform for Experimental VoD Research. In *Proceedings of the 6th IEEE Symposium on Computers and Communications (ISCC'01)*, July 2001.

[9] M. Zink, C. Griwodz, J. Schmitt, and R. Steinmetz. Scalable TCP-friendly Video Distribution for Heterogeneous Clients. In *Proceedings of the Multimedia Computing and Networking 2003 (MMCN'03)*, pages 102–113, January 2003.

[10] The MPlayer project. mplayer. http://mplayerhq.hu, 2005.

[11] P. Halvorsen, T. Plagemann, and V. Gobel. Integrated Error Management for Media-on-Demand Services. In *Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2001)*, pages 621–630, April 2001.

[12] R. Koster, A.P. Black, J. Huang, J. Walpole, and C. Pu. Infopipes for composing distributed information flows. In *Proceedings of the 2001 international workshop on Multimedia middleware (M3W)*, pages 44–47, 2001.

[13] F. Kon, R. Campbell, and K. Nahrstedt. Using dynamic configuration to manage a scalable multimedia distribution system. 24(1):105–123, January 2001.

[14] G. Coulson, G. Blair, M. Clarke, and N. Parlavanzas. The design of a configurable and reconfigurable middleware platform. *Distributed Computing Journal*, 15(2):109–126, 2002.

[15] L. Capra, W. Emmerich, and C. Mascolo. CARISMA: Context-aware reflective middleware system for mobile applications. *IEEE Transactions on software engineering*, 29(10):929–945, 2003.

[16] F. Kon, M. Roman, P. Liu, J. Mao, T. Yamane, L. C. Magalhaes, and R.H. Campbell. Monitoring, Security, and Dynamic Configuration with the dynamicTAO Reflective ORB. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'2000)*, pages 121–143, April 2000.

[17] R. Schantz, J. Loyall, M. Atighetchi, and P. Pal. Packaging quality of service control behaviors for reuse. In *Proceeding of the 5th IEEE International Symposium on Object-Oriented Real-time distributed Computing*, April-May 2002.

[18] P.K. Sharma, J.P. Loyall, G.T. Heineman, R.E. Schantz, R. Shapiro, and G. Duzan. Component-Based Dynamic QoS Adaptations in Distributed Real-Time and Embedded Systems. In *International Symposium on Distributed Objects and Applications (DOA)*, October 2004.

[19] S. Mitchell, H. Naguib, G. Coulouris, and T. Kindberg. A QoS Support Framework for Dynamically Reconfigurable Multimedia Applications. In *in L. Kutvonen, H. König and M. Tienari (eds), Distributed Applications and Interoperable Systems II*, pages 17–30, 1999.

[20] B.D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, and K.R. Walker. Agile Application-Aware Adaptation for Mobility. In *Proceedings of the 16th ACM Symposium on Operating System Principles*, pages 276–287, October 1997.

[21] E. Brewer, R. Katz, E. Amir, H. Balakrishnan, Y. Chawathe, A. Fox, S. Gribble, T. Hodes, G. Nguyen, V. Padmanabhan, M. Stemm, S. Seshan, and T. Henderson. A Network Architecture for Heterogeneous Mobile Computing. *IEEE Personal Communications Magazine*, 5(5):8–24, October 1998.

[22] V. Bharghavan and V. Gupta. A Framework for Application Adaptation in Mobile Computing Environments. In *Proceedings of the 21st International Computer Software and Applications Conference (COMPSAC'97)*, pages 573–579, August 1997.

[23] K. Nahrstedt X. Gu. Distributed Multimedia Service Composition with Statistical QoS Assurance. *IEEE Transactions on Multimedia*, to be published 2005.