Performance Evaluation of a DVB-H Enabled Mobile Device System Model

Lukas Pustina, Simon Schwarzer, Michael Gerharz, and Peter Martini
Institute of Computer Science IV, University of Bonn
D-53117 Bonn, Germany
{gerharz,martini,pustina,schwarzer}@cs.uni-bonn.de

Volker Deichmann
Nokia Research Center, Bochum
D-44807 Bochum, Germany
volker.deichmann@nokia.com

ABSTRACT

In this experience report, we present experiences we have gained in applying performance engineering techniques during the design of a DVB-H enabled handheld device. The modelling methodology we applied uses UML 2.0 to model the system following a strict separation of architectural and behavioural aspects of the systems. From sequence diagrams and composite structure diagrams, a queueing network is generated for the analysis of the system performance. The configuration of the hardware resources and the resource demands is done using the standard SPT-profile. We describe our implementation and its seamless integration into a UML 2.0 CASE tool. Finally, the paper outlines lessons learnt during the design process which may be used to enhance the methodology.

Categories and Subject Descriptors

D.4 [**Computer Systems Organization**]: Performance of Systems—*Design studies; Modeling techniques*; I.6.5 [**Computing Methodologies**]: Simulation and ModelingModel Development[Modeling methodologies]

General Terms

Experimentation, Performance

Keywords

Performance Evaluation, UML, Queueing Networks

1. INTRODUCTION

When designing hardware platforms, inappropriate design decisions have a strong impact on the development costs if they result in the need to re-manufacture prototypes of the envisioned device. A major cause for such re-designs is the discovery of performance bottlenecks during product performance tests. In order to discover these bottlenecks before actually building the prototype, performance engineering is a commonly suggested means. A plethora of performance engineering approaches promise to eliminate the need for costly product re-designs by integrating performance evaluations to the front from the first design steps on.

In this report, we describe experiences we have gained from applying performance engineering methods during the development of a DVB-H enabled handheld device. We outline the approach taken to model and evaluate the system, provide some performance results and summarise lessons learnt during that process. Our goal was to design a general platform for a handheld device which is able to decode terrestrial digital TV programme (DVB-H) and still has sufficient computing resources that common applications such as email, calendar synchronisation etc. may be run in parallel to decoding the DVB-H MPEG-4 video and audio streams.

A major challenge was that building prototypes of such handheld devices is a costly task. Therefore, design decisions needed to be justified carefully from the first design steps on. An important decision with a major impact on the actual implementation was the choice of the processor architecture. In essence, we had to evaluate whether existing one chip mobile phone platforms were already powerful enough to support this application or additional hardware such as a DSP would be required. While further questions, e.g. concerning the influence of the memory architecture, have been studied, our report focuses on this processor issue.

There are many approaches to performance engineering in these early phases of a development process. The majority of these publications propose UML as the modelling framework and suggest to derive performance models from a performance annotated UML system model. To evaluate the performance of the system, prominent approaches such as [24] suggest the use of queueing network models to detect performance bottlenecks in the earliest phases of development when a detailed functional model is not yet available. This matched our challenge closely since a detailed functional model depended on the decision for or against a DSP supported architecture.

Consequently, for the DVB-H design we have assembled a UML based performance engineering framework which relies on queueing network analysis to disclose possible bottlenecks in the system and to thereby substantiate early design decisions. The methodology builds upon work found in the literature and enhances the applied methods where necessary to gain a practical modelling environment.

The rest of this paper is structured as follows. Section 2 reviews related work and provides an overview of relevant existing performance engineering approaches. Section 3 describes in detail the performance scenario we have evaluated. Sections 4 details our framework and outlines its application in the design of a DVB-H

WOSP'07, February 5–8, 2007, Buenos Aires, Argentina.

handheld device. Numerical results gained during the design process are presented in section 5. Lessons learnt during the design process are outlined in section 6. Finally, section 7 presents a summary and ideas for future work.

2. RELATED WORK

Williams and Smith introduced the usage of queueing networks in the context of integrated performance analysis. In their Software Performance Engineering (SPE) approach, UML diagrams are used to specify the model from which a performance model is generated for the analysis [24, 26]. Using UML diagrams for system and performance scenario modelling has become the de-facto standard and several authors presented methods[1, 4, 7, 8, 20, 13, 27, 12, 24, 6]. Since UML does not support the modelling of non-functional hardware and software properties, often the UML Profile for Schedulability, Performance, and Time Specification (SPT profile) [15] is adopted [27, 2, 1, 11].

Methods to derive standard or extended queueing networks from different kinds of UML diagrams are described in [1, 26, 24, 7, 12, 6, 11].

More precisely, the authors of [1] propose the usage of multi-class queueing networks derived from UML models using use case, activity and deployment diagrams. Software and hardware aspects can be modelled with these diagrams and are transformed into a queueing network. In [11] multi-chain queueing networks are used, in such a way that special features of sequence diagrams like alternatives and parallel execution can be modelled. UML 2.0 component diagrams are used to model properties of components which are to be transformed into queueing centers. This approach mainly focuses on software aspects while hardware aspects of the system are not considered.

In contrast to [1, 7, 6, 11], we use the new composite structure diagram of UML 2.0 for hardware modelling. The advantage is that the composite structure diagram supports a hierarchy of diagrams such that an easy refinement of the model during the development process becomes possible.

Moreover, we use sequence diagrams for modelling the behavioural software aspects similar to [7, 6, 26, 27, 11] which use either generic message sequence charts or also UML sequence diagrams.

The specification of the scenario with the help of use case diagrams by specifying the involved workloads is widely used and also applied by [7, 6].

As mentioned above, in our approach the mapping of software to hardware is done with the help of composite structure diagrams by reusing the system components defined in class diagrams as described in [21]. This way, the mapping may be performed graphically in the UML model and developers have to change only one diagram while the changes are automatically propagated to all related diagrams.

We implemented and applied a methodology that automatically derives queueing networks from UML 2.0 diagrams annotated according to the SPT profile. Ideas from different publications, namely the generation of multi-class queueing networks from sequence, class, and use case diagrams, are combined and extended by the usage of the composite structure diagram which can be used to model different software to hardware mappings and enables a refinement model.

3. THE EVALUATION SCENARIO

In this section we outline our task and give a brief introduction into the techniques related to DVB-H.

Digital Broadcasting Video - Handheld (DVB-H) is part of the emerging DVB family of standards [18] for the digital transmission of broadcasts and thus, a successor to analogue TV. DVB-H was designed to deliver TV like content to mobile handhelds and phones. Generally, DVB-H content uses time division multiplexing to transport several channels on the same radio frequency. By periodically using the same time slot for the same channel, the receiving device can reduce the activation of the receiver to these time slots only and thus, reduce the necessary power consumption. The content itself is encoded according to the MPEG-4 [10] standard (this includes the video as well as audio data), but the actual algorithm may be chosen by the content provider.

The *Motion Picture Expert Group standard 4* (MPEG-4) is a family of standards specifying the *encoder and decoder* (codec) algorithm capabilities for video and audio compression, the container format for transmission, multiplexing and synchronisation of audio and video as well as several other standards like testing procedures, reference hardware etc. The audio content is encoded using *Advanced Audio Coding* (AAC) which is basically a successor of the widely used MP3 codec and builds on top of the *modified discrete cosine transformation* (MDCT). The video data is encoded by slicing the continuous video signal into discrete pictures called frames in a determined frequency. Common frame frequencies are 24, 25 or 30 frames/sec. Each frame is separated into so-called *Macro Blocks* (MB) which are encoded individually per each frame. MPEG-4 defines three frame types. *Intra-coded frames* (I-frames) are standalone pictures and, similar to JPEG DCT encoded, quantised, and finally variable-length encoded. In *forward Predictive frames* (P-frames) each MB is predicted by motion vectors from preceding I- or P-frames. With *Bi-directional predicted frames* (B-frames) each MB is predicted from both, a preceding I- or P-frame and a succeeding I- or P-frame. The prediction errors of B- and P-frames are encoded in the same way as I-frames. The sequence of I-, P-, and B-frames is called *Group of Pictures* (GoP) and can vary according to the user requirements.

Our task was to model a mobile device capable of decoding an MPEG-4 video and audio stream while still supplying enough performance reserves to allow additional working like writing emails or reading text messages. We used UML 2.0 as the modelling language as commonly suggested by the literature and because of its wide spread use. In the beginning we set 3 requirements for our modelling approach in order to allow for the practical application. These requirements are (1) support of iterative system composition and non-invasive integration of a performance model into the functional system model in order to enable developers to use their modelling approaches and allows all non-performance engineers to work unaffected without thinking of the performance model. (2) coherent model which supports the reuse of already modelled parts of the system (e.g. classes as objects/instances). (3) a strict separation of the architecture and behaviour of the system which allows for an easy exchange or modification of the architecture without remodelling the behaviour and thus, enables an easy comparison of different design alternatives.

4. THE PERFORMANCE ENGINEERING APPROACH

In this section, we present our system modelling approach (sec. 4.1) followed by the incorporation of performance aspects into this model. Section 4.2 briefly presents the transformation algorithm we used to generate multi-class queueing networks which are a widely accepted performance estimation means. In order to keep the model easily understandable, we only present high level diagrams. Apart from the configuration shown in this paper, our study

also included components like memory, air interface, display etc. as well as transmission error, a detailed MPEG decoding behaviour et al. Details may be found in [21].

4.1 Modelling

The UML model of the MPEG-4 enabled mobile device is split into two logical parts. First, the functional system aspects are described in the *system model* (cf. sec. 4.1.1) and then, the performance aspects are specified in the *performance model* (cf. sec. 4.1.2) using annotations in the system model. Section 4.1.3 describes how these models can be used to specify a specific performance evaluation scenario.

4.1.1 System Modelling

Today's handheld devices are often based on a combination of a *Multipurpose Processor Unit* (MPU) and a *Digital Signal Processor* (DSP). A very popular incarnation of this system layout is the *Texas Instruments' OMAP* platform (OMAP). It combines an ARM based MPU with a DSP. These two components are connected and operate on the received data according to the decoding algorithm driven by an operating system or application. To model such a system, three steps are necessary. The components of the system, their architectural layout, and the algorithms running in the system have to be specified. This separation into component, architecture, and behaviour modelling is also suggested in the literature and in [9]. Figure 1 shows the system components of the current system abstraction level. We used a class diagram to model these components in UML. The `MPU` and the `DSP` classes specify the processing hardware components. The `System` class corresponds to the operating system and the application running the decoding process. The video frame decoding and audio block decoding are modelled by the `VideoDecompression` and `AudioDecompression` classes. The `DCT` class is an abstraction of the MDCT and DCT used by the video and audio decoder representing the most demanding part of the decoding process. We also used the class diagram to specify the public methods of all classes (see below).

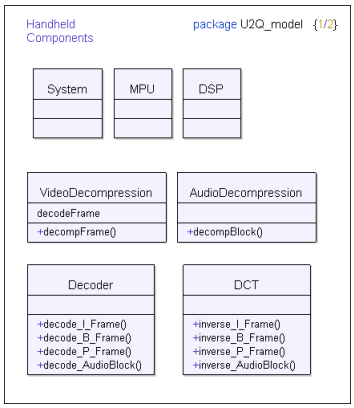

Figure 1: Class diagram of handheld device components

The authors of [11] suggest to use the component diagram for modelling the topology. We basically adapted their approach, but employ the similar composite structure diagram, because it allows us to semantically reuse classes as instances as its elements. Another benefit of the composite structure diagram is its ability to decompose classes into its internal parts. By specifying the internal structure of a class which is part of a composite structure diagram using another composite structure diagram, a hierarchy of composite structure diagram evolves. This approach can be used to iteratively refine the system model towards a sophisticated model. Figures 2 and 3 show the internal structure of the handheld device and the `MPU` component. The connection between the `MPU` and `DSP` classes in figure 2 enables both to exchange data. In order to analyse the impact of the `DCT` running either on the `MPU` or `DSP` it is only necessary to use another composite structure diagrams refining either the `MPU` or the `DSP` by the `DCT` (as figure 3 depicts for the `MPU` case).

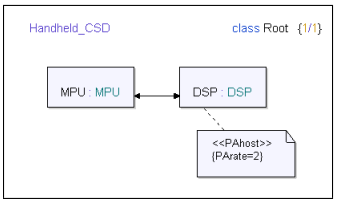

Figure 2: Root CSD of handheld device

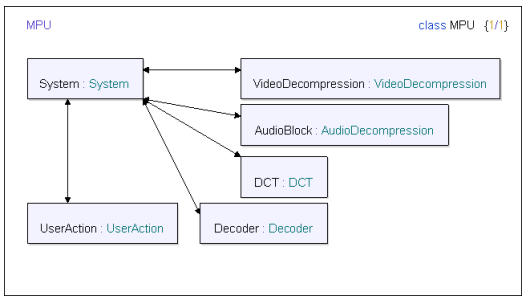

Figure 3: Composite Structure Diagram of MPU

Figure 4 presents a tree showing the composite structure diagram hierarchy. The top level diagram (`Handheld_CSD` in this case) we call the *root composite structure diagram* (root CSD). From the root CSD all other composite structure diagram are reachable in the hierarchy.

Figure 4: Composite Structure Diagram hierarchy

The final part of the system model, the dynamic behaviour, is modelled using sequence diagrams. Again, the class definitions of the system components may be reused as class instances exchanging messages in this diagram type. The messages are directly related to the public methods of the particular classes. Figure 5 presents a simplified sequence diagram corresponding to the video decoding process. Depending on the frame type (cf. sec. 3) the DCT is run with different performance demands (cf. sec. 4.1.2). A similar diagram may be shown for audio decoding.

4.1.2 Performance Modelling

This section shows how to incorporate performance aspects, i.e. performance demands and performance capacities, into the system model in a non-invasive manner such that the system model

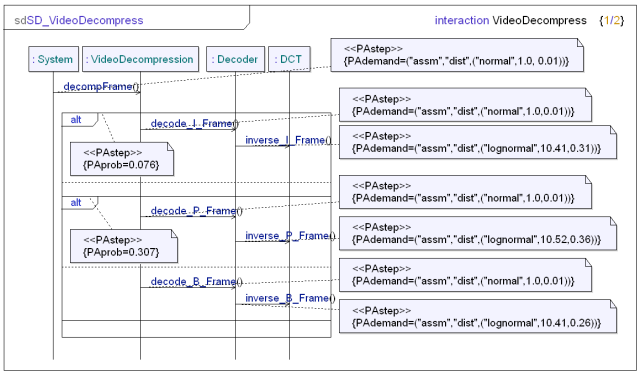

Figure 5: Sequence Diagram of Video Decompression

remains usable without the performance model (see the requirements in sec. 3). As widely proposed, we use the "UML profile for Schedulability, performance, and time" (SPT-Profile) for performance annotation. Even though this profile was designed for UML 1.x, it can be easily used in UML 2.0 as well or easily modified to fit our needs.

Two step are necessary to describe the performance aspects of a system. First, each component's capacity has to be modelled. Second, the resource demand of the dynamic behaviour has to be specified.

To model the component capacities, the SPT-Profile tagged value `PArate` is used as part of the `<<PAhost>>` annotation as a relative speed value. In case of a `PArate=1` the annotation may be omitted. Due to the fact that the capacity is not a functional aspect of a component, we use the composite structure diagram for capacity annotations rather than the class diagram. Thus, it is possible to not only change the system components topology, but also the performance of its components by exchanging the particular composite structure diagram as described in section 4.1.1. Thus, the whole architecture (topology and performance capacities) can be exchanged at once allowing for an easy evaluation of design alternatives. Note, that using the composite structure diagram for the capacity modelling violates requirement (3) (cf. sec. 3 and sec. 6).

This annotation can be seen in figure 2. If modifying the relative speed of the `MPU`, it is sensible that as a consequence the refining components (e.g. the `DCT`) should result in a speed up as well. Therefore, a modification of a single component's capacity should automatically propagate to its sub-components. In order to obtain absolute speeds for each component in the context of the whole system, the `PArate` values are multiplied along their path through the composite structure diagram hierarchy. Additionally, the scheduling policy can be specified with the `PAschdPolicy` tagged value and is set to FIFO by default.

During the second step of the performance modelling, each message exchange representing the execution of an action in the receiving object is annotated with the SPT-profile `PAdemand` tagged value as part of the `PAstep` annotation. Figure 5 depicts this annotation.

4.1.3 Performance Scenario

The engineer's perspective on the model changes during the development process due to different aspects of the system to analyse. Therefore, it is crucial to allow the developer to focus on and to analyse only certain parts of the system. This includes architectural as well as behavioural aspect.

A *performance scenario* is a set of use cases and corresponding architectural system parts. Each use case is represented by a sequence diagram. Architectural system parts are described by composite structure diagrams hosting the participating components (cf. sec. 4.1). Additionally, the performance scenario contains the workload definition for each use case. These workloads are defined by the `<<PAopenLoad>>` or `<<PAclosedLoad>>` annotation as suggested by the SPT-Profile. Since these two workloads annotations only specify stochastic distributions, we introduce a third, non SPT-Profile conform workload definition called `<<PAtraceLoad>>` which allows us to specify a trace file as workload that triggers the arrival of jobs according to a time table gathered from simulations or measurements. In this way, we can easily switch from assumed to realistic arrival rates without modifying the system (cf. sec. 4.2).

Figure 6 shows a use case diagram with two use cases (`Audio-Decompression`, `VideoDecompression`) and their workloads. Each use case is described by a sequence diagram, e.g. see figure 5 for the `VideoDecompression` use case.

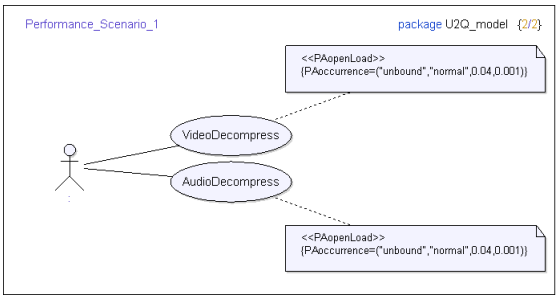

Figure 6: Use Case Diagram of Performance Scenario

The architecture is described by composite structure diagrams in a sub-tree of the hierarchy that contains all participating components. The *processing composite structure diagram* (processing CSD) is the root node of this sub-tree to which all components are mapped during the queueing network generation (cf. sec. 4.2). For example, if the engineers are only interested in the performance of the algorithm parts running on the MPU they choose the MPU composite structure diagram as the corresponding processing CSD.

4.2 Transformation to Queueing Networks

In this section, we briefly present the algorithm that combines the system and performance model with the performance scenario and generates a multi-class queueing network representing the use cases of the particular performance scenario. This queueing network may be evaluated analytical or by simulation depending on its characteristics. If the queueing network is in product form and thus, fulfils the BCMP rules[3], an analytical evaluation is possible. Otherwise, a simulative evaluation must be employed. In [21], we presented a detailed description of the transformation algorithm.

The transformation algorithm works in 4 steps as depicted in figure 7. The first three steps are performed for each sequence diagram specifying a single use case in the performance scenario while step 4 combines the results to the final queueing network.

In step 1, the *chain of actions* is derived from the actions initiated by a message reception in a sequence diagram. Thus, the flow of messages and their receiving objects define the chain of actions. Note that interaction frames need a separate processing. The algorithm only evaluates the `alt` and `loop` interaction frames, because they influence the flow of actions and are compatible with queueing networks. The `opt` interaction frame can be modelled with an `alt` frame with only one branch. The `par` interaction frame cannot be transformed into a queueing network equivalent representation, because queueing networks do not support forks or joins of jobs.

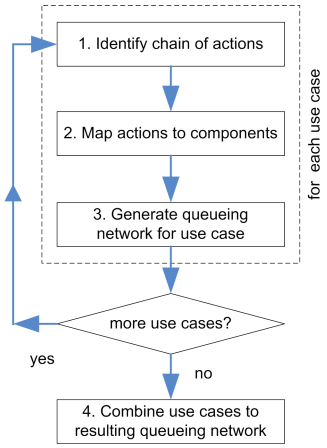

Figure 7: The four steps of the transformation algorithm

They are necessary to simulate the semantic meaning of the `par` interaction frame. The remaining interaction frame types do not influence the execution logic. Since the processing of interaction frames is delegated to step 3 (see below), the chain of actions also contains representations for `alt` and `loop` interaction frames.

In step 2, the components that perform the actions as identified by step 1 are mapped to the processing CSD according to the architectural hierarchy. The components of the processing CSD are the components executing the action in the current view of the system. During this step, the performance capacities of the components are calculated as described in section 4.1. As a result of step 2, the chain of actions now only consists of processing resources in the same sequence as the messages flow in the underlining sequence diagram. Each step is assigned the resource demand associated with the corresponding message.

Step 3 transforms this chain of actions into a single queueing network representing the use case. This includes the definition of the workload and the queueing centers as well as unfolding the flow of actions together with the `alt` and `loop` interaction frame representations specified in the chain of actions. Due to the mapping of message receivers to processing CSD components, different receivers might be mapped to the same component and thus, to the same queueing center. This leads to an implicit loop, because the jobs have to revisit the corresponding queueing center. Together with explicit loops (defined by the `loop` interaction frame) and alternatives (`alt` interaction frame) multiple job classes are used to distinguish between revisits. After a job of a specific class was processed, its job class is modified according to the next queueing center in the network to visit.

Steps 1-3 are repeated for each use case and its sequence diagram. Finally, step 4 combines all single queueing networks to the *resulting queueing network* which may be further analysed. This queueing network operates with multiple workloads (one for each use case) and several job classes to distinguish between revisits. This is necessary, in order that the queueing centers can differentiate between the use cases and process the incoming jobs of different workloads with the according service times. Therefore, the job classes assigned in step 3 are mapped to unique job classes of the resulting queueing network. Figure 8 depicts the resulting queueing network for the MPEG decoding scenario incorporating the `AudioDecompression` and `VideoDecompression` use case.

4.3 Implementation

This section gives a brief overview of the implementation of the transformation algorithm and its integration into the Tau G2 UML 2.0 CASE-tool. Figure 9 depicts the tool structure.

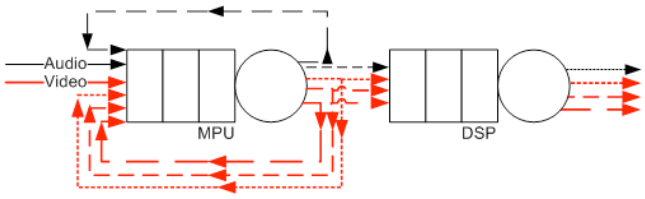

Figure 8: Resulting queueing network for Audio and Video use cases

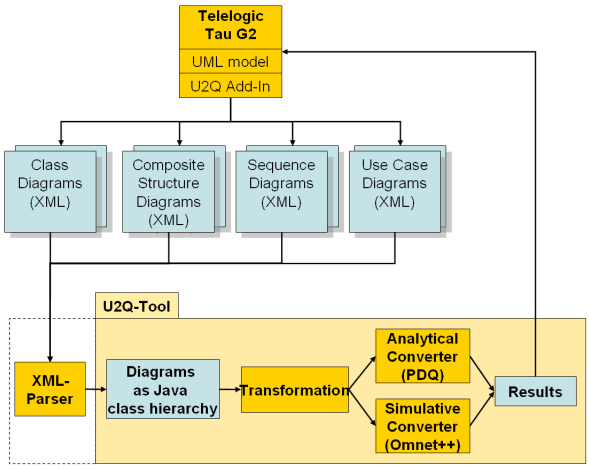

Figure 9: U2Q-Tool overview

The UML diagrams are modelled in Telelogic Tau G2 [25] which stores its data in XML files. These file are read into a proprietary Java class hierarchy by a XML-parser. Our *UML to Queueing Networks* (U2Q) tool operates only on this class hierarchy and thus, enables an easy adaption for other CASE tools by simply replacing the XML-parser. The transformation algorithm supports analytical as well as simulative evaluation. Currently, our tool generates input for the analytical tool PDQ[19] and the general purpose simulator Omnet++[16]. Both tools were enhanced where necessary. By separating the queueing network transformation from the actual evaluation tool input generation, it is easy to extend U2Q by new evaluation tools.

The transformation process can be configured and started by an add-in directly from Telelogic Tau. This allows the developers to run the evaluation without changing the application. In a configuration profile the use cases and composite structure diagrams, the intended queueing network evaluation tool, and the output directory can be specified. Moreover, this add-in is able to analyse the evaluation output and to write it back into the diagrams. For example, the utilisation of a hardware component and the mean residence time are presented as a comment of the corresponding object in the composite structure diagram. Detailed simulation results can be viewed in a separate window and are saved in a HTML file. Figure 10 shows the profile configuration dialogue as an example of the add-in.

5. PERFORMANCE ANALYSIS

In this section, we show the performance analysis steps taken

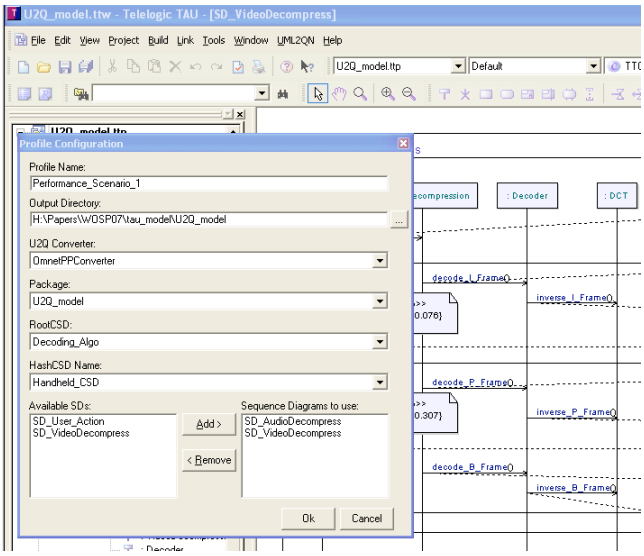

Figure 10: Screen shot of the TAU G2 Add-In

for the described scenario (cf. sec. 3). We considered two architectures. Architecture (1) consists only of an `MPU` running all parts of the decoding algorithm. In architecture (2) we add a `DSP` component which was assigned the computing intensive algorithm parts. Both architectures were evaluated by simulation with arrival and service times modelled stochastically. Additionally, we ran the simulation of architecture (2) in the trace based mode as well in order to compare the stochastic model with real world values.

Section 5.1 describes how we obtained the necessary performance figures for specifying the performance model. In section 5.2 we present the results of our simulations.

5.1 Input Values

In order to gain reasonable performance values, it is necessary to use appropriate figures for the resource capacities and the resource demands. The goal of our analysis was to gather information about the resource demand of an MPEG decoder running on an OMAP platform. As mentioned in section 3, one goal of our analysis was to decide whether the MPU offers sufficient computing resources or whether the DSP functionality would be needed. Therefore, it was sensible to estimate the resource demands of the decoding process running on an ARM MPU only, first.

For this purpose, we measured the resource demands of decoding an MPEG stream on a real ARM9TDMI. First, we recorded a DVB-T stream (using MPEG-2 for video and AAC for audio) of approx. 30 min and transcoded it into an H.264 encoded video stream with PAL QCIF resolution (176x144 pixels) and the GoP "IBBPBBPBBPBBP" (cf. sec. 3). The audio stream was kept the same as in the original DVB-T stream. In a second step, we modified the open-source video playback tool *mplayer*[14] used by the OpenZaurus[17] project, a FreeBSD distribution for the ARM based Sharp Zaurus device. We extended mplayer to support measurements for the decoding times of each I-, P-, and B-frame as well as for each audio packet. Table 1 shows a summary of the measurement figures. The values for the audio and video decoding were measured independently. For this purpose, we configured mplayer to only decode one of the streams and to ignore the second. The measured values show a significant higher processing time for audio packets. This is due to the fact that the audio stream was kept in the original quality while the video quality was reduced.

	Min.	Median	Mean	0.975 Qu.	Max.
Audio	2272	73794	85274	237768.4	712103
Video	14610	35294.5	37510	67996.12	218800

Table 1: Measured performance figures for decoding video and audio on ARM in usec

From our measurements we observed several very large decoding times for both, video and audio frames. These values are likely due to operating systems tasks running in parallel. Therefore, for our analysis we considered the top 2.5% of the decoding times as outliers.

For modelling the queueing centers we analysed the distribution of the decoding times. Using a maximum-likelihood-fitting, we concluded that the decoding times are roughly lognormally distributed. Figure 11 depicts the pdf of the video frame decoding times and a fitted lognormal distribution with the parameters $\mu = 10.45, \sigma = 0.31$. The decoding times of the different frame types can well be distinguished by the three peaks in the sample pdf. Nevertheless, the lognormal distribution provides a good approximation of the distribution. Similar graphs may be drawn for the distributions of the single frame types as well as the audio packets.

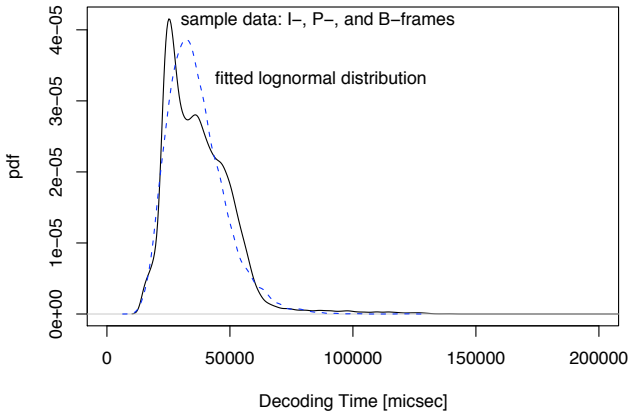

Figure 11: Density of frame decoding times and fitted lognormal

Table 2 presents all distribution parameters used for the stochastic evaluation.

	Mean	StdDev
I-Frame	10.41	0.31
P-Frame	10.52	0.36
B-Frame	10.41	0.26
Audio	11.06	0.74

Table 2: Parameters for service time modelling using log.norm. distributions

For the DSP, an implementation of the MPEG decoding algorithm was not available. Since the goal of the methodology was to gain performance estimates early, but without starting actual implementational work, we estimated the execution time of DSP code from figures found in the literature. The authors of [5] break down the MPEG decoding algorithm into 5 major functional parts and provide an analysis of how much time is spent in each functional

part. We used this analysis to determine the fraction of time spent in those functional parts in order to delegate their computation to the DSP. For example, DCT accounts for 59% of the computational complexity, interpolation 12%, and motion estimation is performed 10% of the time.

In [23], figures are presented for the speed-up in execution time that a DSP optimised implementation may be expected to achieve compared to a pure implementation on the MPU. According to these figures, DCT may be expected to execute 4.1 times faster, interpolation 7.3 times faster and motion estimation may be expected to accelerate by a factor of 5.2.

5.2 Performance Results

As mentioned before, the video and audio stream decoding times were measured separately, because the OMAP MPU is not capable to decode both streams in real time. The same result was given by our simulation of architecture (1) as expected.

The second architecture incorporated a DSP and thus, allowed us to distribute parts of the decoding algorithm to this special purpose processor. This has two advantages. First, the MPU is relieved and second, the optimised signal processing routines leads to significant decrease of computation times. According to the results of [23] we moved the complex decoding parts to the DSP step-by-step. We started with the inverse DCT (59% of computation time) for the audio and video decoding which still led to an overload of the MPU. Only after moving the interpolation (12%) and the motion compensation (10%) to the DSP, too, the MPU load dropped low enough to allow the user to perform background activities as required (cf. sec. 3). During this modelling phase we distinguished between I-, B-, and P-frames. The probability for each frame type directly follows from the GoP (cf. sec. 5.1) which is $p_I = 1/13$, $p_B = 8/13$, and $p_P = 4/13$ in this scenario. Table 3 presents the final results with all three mentioned algorithm parts running on the DSP which accounts for 81% of the computational complexity.

Sim. run	**MPU**	**DSP**
stochastic	63%	57%
trace-based	62%	56%

Table 3: Utilisation of MPU and DSP in architecture (2)

It can be clearly seen that both processors have enough resources to perform additional background tasks. Further, it is sensible to invest into porting parts of the decoding algorithms to the DSP.

In addition, we ran the simulation of architecture (2) with the trace files generated by our measurements. In these trace files the frame type, the processing time, and the frame number are saved for each processed frame. Thus, we could directly compare the trace based simulation results with the performance figures from the stochastically modelled simulation which showed to be an appropriate approximation (cf. tab. 3).

6. LESSONS LEARNT

In this section, we want to share our experiences with the presented methodology applied to the concrete example.

The presented system modelling approach proved to be applicable and well suited for the given task to model a DVB-H enabled mobile phone. The intuitive separation of aspects into components, architecture, and behaviour model could be strictly kept during all stages of the system design. Since the chosen UML Case tool Tau G2 supports the direct linking of class diagrams with composite structure diagrams, it was possible to iteratively decompose the system as suggested by our approach.

For specifying the resource capacities in the performance model (cf. sec. 4.1.2) we dropped the requirement of a strict separation of architecture and behaviour (requirement (3), cf. sec. 3). Since the capacities are rather part of the architectural system design, we decided to move their modelling to the composite structure diagrams instead of the class diagrams. Even though this violates the required separation of component and architecture modelling, it enables a simple exchange and comparison of architectures.

The SPT-Profile annotations for modelling resource demands are easy to use, but have two drawbacks. First, the specification of service times is very technical. Designers unfamiliar with queueing networks still need to understand queueing theory in order to choose the appropriate service time distributions (and how to estimate them from already known data). This applies for the specific distribution parameters as well. Second, the actual annotation values lack any information about their composition. For example, distributing the video decoding service time by 19% to the MPU and by 81% to the DSP (cf. sec. 5.2) results in manual calculations and annotations. The informaion how these values were created is lost as there is no direct description. This makes the performance model hard to understand.

The performance input values (capacities and demands) are an integral part of the whole performance evaluation and thus should be as close to reality as possible. However, obtaining these values is very difficult, if the system to model has no real world implementation. During our research we studied a plethora of papers and technical reports in order to gain sensible input values for our model. Since this approach did not lead to satisfying results, we developed our own testbed for measurements as described in section 5.1. This testbed included a fully functional OMAP board running a Linux distribution and a patched version of mplayer, in other words an early prototype. We suppose that the lack of appropriate performance input values is inherent to all performance modelling approaches and should be further investigated. A database with performance figures for common components and algorithm demands could be a first step towards a practicable solution accepted by developers.

The integration of our U2Q tool into the UML CASE tool proved to be an elegant and sophisticated way to enable a practicable employment of our methodology. The designers do not need to change to another application, but only configure the performance scenario and receive the values in a detailed report. The direct write back of basic performance results such as the utilisation and the throughput into the model itself visualises bottlenecks. A further enhancement could be to tag highly stressed components in colour. Additionally, the detailed report could be modified to show only metrics of interest.

7. SUMMARY & FUTURE WORK

In general, the methodology we presented throughout this paper proved to be well suited for the given task. The modelling approach is close to design methodologies already established and thus easily adoptable by designers and developers. It builds upon UML 2.0 and implements a strict separation of architectural and behavioural aspects of the system. The former are modelled using class and composite structure diagrams while the latter are modelled using sequence diagrams. Performance annotations are included using the standard SPT-profile.

The automatic generation and evaluation of a queueing network directly from the development tool enables an easy application of performance analysis of system designs. The seamless integration hides the technical details of queueing networks from the users. However, it is a very complicated task to gather realistic perfor-

mance input values which are essential for reasonable evaluations. In order to get the developers' acceptance for a performance engineering enhanced design methodology, its usage must be as less interfering with their routine work methods as possible. This also includes obtaining performance input values. This should be part of further research work.

8. REFERENCES

[1] S. Balsamo and M. Marzallo. Performance evaluation of uml system architectures with mutliclass queueing network models. In *WOSP*, 2005.

[2] S. Balsamo, M. Marzolla, A. D. Marco, and P. Inverardi. Experimenting different software architectures performance techniques: a case study. In J. J. Dujmovic, V. A. F. Almeida, and D. Lea, editors, *WOSP*, pages 115–119. ACM, 2004.

[3] F. Basket, K. Chandy, R. R. Muntz, and F. Palacios. Open, closed, and mixed networks of queues with different classes of customers. *Journal of the ACM*, 1975.

[4] S. Bernardi, S. Donatelli, and J. Merseguer. From uml sequence diagrams and statecharts to analysable petri net models. In *WOSP '02: Proceedings of the 3rd international workshop on Software and performance*, pages 35–45, New York, NY, USA, 2002. ACM Press.

[5] J. Chaoui, S. de Gregorio, J.-P. Giacalone, J. Webb, and Y. Masse. Open multimedia application platform: Enabling multimedia applications in third generation wireless terminals trough a combined risc/dsp architecture. *Acoustics, Speech, and Signal Processing, 2001. Proceedings. (ICASSP)*, 2001.

[6] V. Cortellessa, M. Gentile, and M. Pizzuti. Xprit: An xml-based tool to translate uml diagrams into execution graphs and queueing networks. In quest2004 [22], pages 342–343.

[7] V. Cortellessa and R. Mirandola. Deriving a queueing network based performance model from uml diagrams. In *Workshop on Software and Performance*, pages 58–70, 2000.

[8] V. Cortellessa and R. Mirandola. Prima-uml: a performance validation incremental methodology on early uml diagrams. *Sci. Comput. Program.*, 44(1):101–129, 2002.

[9] ETSI. *UML Profile for Communicating Systems (draft)*. ETSI, 2005.

[10] ISO. Iso/iec 14496. http://www.dvb-h.org, 1998.

[11] A. D. Marco and P. Inverardi. Compositional generation of software architecture performance qn models. In *WICSA*, pages 37–46. IEEE Computer Society, 2004.

[12] M. Marzolla and S. Balsamo. Uml-psi: The uml performance simulator. In quest2004 [22], pages 340–341.

[13] J. Merseguer and J. Campos. Software performance modeling using uml and petri nets. In M. Calzarossa and E. Gelenbe, editors, *MASCOTS Tutorials*, volume 2965 of *Lecture Notes in Computer Science*, pages 265–289. Springer, 2003.

[14] MPlayer. http://www.mplayerhq.hu.

[15] OMG. *UML Profile for Schedulability, Performance, and Time Specification: Version 1.0*. Object Management Group, 2003.

[16] Omnet++ – general purpose network simulator. http://www.omnetpp.org.

[17] OpenZaurus. http://www.openzaurus.org.

[18] D.-H. org. Dvb-h homepage. http://www.dvb-h.org.

[19] Pretty damn quick (pdq). http://www.perfdynamics.com.

[20] R. Pooley and P. King. The unified modeling language and performance engineering. In *IEE Proceedings — Software.*, 1999.

[21] L. Pustina, V. Deichmann, M. Gerharz, P. Martini, and S. Schwarzer. Performance aware design of communication systems. In *Proceedings of LCN 2006*, 2006.

[22] *1st International Conference on Quantitative Evaluation of Systems (QEST 2004), 27-30 September 2004, Enschede, The Netherlands*. IEEE Computer Society, 2004.

[23] K. Ramkishor and V. Gunashree. Real time implementation of mpeg-4 video decoder on arm7tdmi.

[24] C. U. Smith and L. G. Williams. *Performance Solutions, A Practical Guide to Creating Responsive, Scalable Softwarre*. Addison-Wesley Pearson Education, 2001.

[25] Telelogic tau g2 homepage. http://www.telelogic.com.

[26] L. G. Williams and C. U. Smith. Performance evaluation of software architectures. In *WOSP*, pages 164–177, 1998.

[27] J. Xu, C. M. Woodside, and D. C. Petriu. Performance analysis of a software design using the uml profile for schedulability, performance, and time. In P. Kemper and W. H. Sanders, editors, *Computer Performance Evaluation / TOOLS*, volume 2794 of *Lecture Notes in Computer Science*, pages 291–307. Springer, 2003.