# A Practical Approach for Performance-Driven UML Modelling of Handheld Devices – A Case Study

Lukas Pustina [a,*], Simon Schwarzer [a], Michael Gerharz [b], Peter Martini [a], Volker Deichmann [c]

[a] *Institute of Computer Science IV, University of Bonn, D-53117 Bonn, Germany*

[b] *FGAN-FKIE, D-53343 Wachtberg, Germany*

[c] *Nokia Siemens Networks, D-40472 Düsseldorf, Germany*

**Abstract**

In this article, we present a performance engineering enhanced modelling methodology for designing embedded devices and describe the experiences we have gained in applying this methodology during the design of a DVB-H enabled handheld device. The methodology uses UML 2.0 to model the system following a strict separation of architectural and behavioural aspects of the system. For this purpose we employ the new composite structure diagram and show its advantages over already established approaches. This methodology specially aims on an easy application by non performance experts. From the model, a multiclass queueing network is generated for the analysis of the system performance. The configuration of hardware resources and resource demands is done using the standard SPT Profile which is extended where necessary. This makes queueing theory accessible to system designers even if they are not familiar with the underlying mathematics. In this way the acceptance of developers to use performance engineering in their daily work is increased. Special attention has been put on an easy evaluation of design alternatives. We describe our implementation and its seamless integration into a UML 2.0 CASE tool.

*Key words:* Modelling, UML, Performance Engineering, Performance Evaluation, Multiclass Queueing Networks, SPT, MARTE, Composite Structure Diagram

* Corresponding author.

*Email addresses:* pustina@cs.uni-bonn.de (Lukas Pustina), schwarzer@cs.uni-bonn.de (Simon Schwarzer), gerharz@fgan.de (Michael Gerharz), martini@cs.uni-bonn.de (Peter Martini), volker.deichmann@nsn.com (Volker Deichmann).

## 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 already before building prototypes, 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 (1) we described a modelling methodology and the experiences we had gained from applying performance engineering methods during the development of a DVB-H enabled handheld device. We outlined the approach taken to model the system, the transformation of the model into a multiclass queueing network as well as its evaluation, and provided performance results. As an example for illustrating the methodology, we showed how to apply the methodology in order to design a general platform for a handheld device which has been able to decode terrestrial digital TV programme (DVB-H) and still had sufficient computing resources for background tasks.

In this article, we show that by using our methodology, the system designers only need to agree on which functional components the system will consist of, but postpone the decision whether they will be implemented in hardware or software. Our methodology allows the designers to examine different deployment architectures while leaving the architectural decomposition and dynamic behaviour unchanged. We extend the description of the algorithm by pseudocode and demonstrate all relevant aspects along the DVB-H scenario. The evaluation explicitly covers background tasks by modelling a worst-case scenario for sending and receiving e-mails. We discuss the short comings of the SPT Profile and show how they are addressed by the upcoming MARTE Profile or what is still missing.

Building prototypes of handheld devices is a costly task. Therefore, design decisions need to be justified carefully from the first design steps on. An important decision with a major impact on the implementation is the choice of the platform. In essence, it is important to evaluate whether existing one chip handheld platforms are already powerful enough to support a specific application or additional hardware such as a DSP is required. While further questions, e.g.,, concerning the influence of the memory architecture, have been studied, this article focuses on this processor issue.

There are many approaches to performance engineering in the 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 perfor-

mance annotated UML system model. Concerning the performance analysis, different kinds of performance models have been proposed, e.g., execution graphs used in (2; 3; 4; 5), petri nets utilised in (6; 7), (extended) queueing networks instrumented in (8; 2; 3; 4; 9; 5; 10), or layered queueing networks (LQN) (11; 12). The authors of (13) give an overview of different methodologies, categorised amongst others by the kind of performance model used. We follow the suggestion of using (extended) queueing networks to evaluate the performance of the system and detect performance bottlenecks in the earliest phases of development when a detailed functional model is not yet available. This meet our challenge closely, since a detailed functional model depends on the decision for or against a DSP supported platform.

A major benefit of the UML based approach is that queueing theory is hidden from the developer. In this way, the power of queueing network theory and thus, performance engineering, becomes available to system designers unfamiliar with the details of this theory.

The rest of this article is structured as follows. Section 2 describes the performance scenario we evaluate. Section 3 details our framework and outlines its application in the design of a DVB-H handheld device. We describe the exploration of different deployment architectures which is eased by using composite structure diagrams. In section 4 we describe the algorithm that transforms the model into a multiclass queueing network. All steps of the algorithm are presented in pseudocode along the case study. Numerical results gained during the design process are presented in section 5. Finally, section 6 summarises the article and outlines possible future work.

## 2 The Evaluation Scenario

We evaluated our methodology with a scenario where we analysed the impact of a DVB-H enhancement to a handheld device. This section presents an overview of a reduced version of the evaluation scenario that we use throughout this article to demonstrate the methodology. A brief introduction into the techniques related to DVB-H is given.

*Digital Broadcasting Video - Handheld* (DVB-H) is part of the emerging DVB family of standards (14) for the digital transmission of broadcasts and thus, a successor to analogue TV. DVB-H was designed to deliver TV-like content to handhelds and mobile 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 power consumption. The content itself is encoded according to the MPEG-4 (15) standard (this includes the

video as well as audio), 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 to 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, and 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 pictures, 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. According to (16) the decoding process of an MPEG video frame basically works in four steps. First, the bit stream is variable length decoded (`Decode`), the second step is the inverse interpolation (`Interpolation`), the third step applies the inverse DCT (`DCT`), and the fourth step is the motion compensation (`MoComp`). We use these four steps as the main software components in this article.

In order to allow for the impact of decoding a DVB-H stream on ”regular” applications, we also modelled a use case consisting of sending and receiving e-mails. This use case was executed in parallel to the decoding processes.

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 Open Multimedia Applications Platform* (OMAP) (17). It combines an ARM based MPU with a DSP. These two components exchange data and program code via the memory subsystem, e.g., the MPU loads a DSP program into the DSP to execute. These components are connected via the AMBA bus, a high performance bus which allows for priority based scheduling, burst requests, split transactions etc. During our study we had access to an OMAP developer board to run software and to make measurements. Where possible, we also used a real mobile phone to gather performance data.

## 3 The Performance Engineering Approach

In our example, we modelled a handheld device capable of decoding an MPEG-4 video and audio stream while still supplying enough performance capacities to allow additional work like writing e-mails or reading text messages. The usage of UML as modelling language in the context of performance analysis has been proposed by several authors (8; 6; 4; 18; 19; 7; 11; 9; 3; 5) and thus, may be considered the de-facto standard. Therefore, we adopt the idea of generating performance models from a set of UML diagrams for performance analysis in early stages of development. UML 2.0 enhances the semantics of existing diagrams and provides new diagram types. Especially, sequence diagrams offer new features and the new composite structure diagram can be used to model the interconnection of instances.

In the following, the *system architecture* describes the components decomposition and the dynamic behaviour. In contrast, the *deployment architecture* specifies the processing resources and assigns them components to host and execute. Further, it specifies the topology of interconnections between the components.

In the beginning, we set three requirements for our modelling approach in order to ensure the practical applicability of our methodology. These requirements are (1) support of iterative system composition and non-invasive integration of performance annotations into the system model in order to enable developers to use their modelling approaches and allow all non performance engineers to work unaffected without bothering with the performance annotations. (2) a 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 system and the deployment architecture. In this way, an easy exchange or modification of the deployment architecture is possible without re-modelling the behaviour. This, in turn, enables an easy comparison of different platform design alternatives. We put special emphasis on this requirement, because in early stages of development it is crucial to explore different platform designs. In these stages it is often uncertain whether a component will be implemented in hardware or software.

Following these requirements, the UML model of the MPEG-4 enabled handheld device is split into two logical parts. First, the functional system aspects are described in the *system model* (cf. sec. 3.1) and second, the performance aspects are specified by *performance annotations* (cf. sec. 3.2). Using only annotations, the performance aspects are integrated in a *non-invasive* manner in the sense that the system model remains reasonable without the performance annotations. Following this clear separation, it is possible to maintain the system model independently from the performance aspects. Section 3.3 describes how the model and the performance annotations can be used to specify and analyse a specific *performance evaluation scenario* in a straightforward way without the necessity to modify the system model or the performance annotations. This is achieved by defining a set of use cases (the

scenarios of interest) and workloads for each scenario. Furthermore, by specifying which part of the system to analyse, it is possible to evaluate the performance of the whole system or of subsystems only. The specification of the performance scenario and the corresponding workloads with the help of use case diagrams is a widely used and accepted practice (4; 5; 8).

In order to keep the model easily understandable, we only present high level diagrams. Apart from the configuration shown in this article, our study also included components like memory buses, air interface, display as well as transmission errors, a detailed MPEG decoding behaviour etc.

### *3.1 System Modelling*

In our scenario we investigated an OMAP board with an MPU and 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 (20).

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`, `Memory`, and `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 classes `Decoder`, `Interpolation`, `DCT`, and `MoComp` represent the most demanding parts of the MPEG decoding algorithm which are executed for each received frame. The class `DCT` is used as an abstraction for the MDCT and DCT used by the video and audio decoder (cf. sec. 2). We also used the class diagram to specify the public methods of all classes which will be reused.

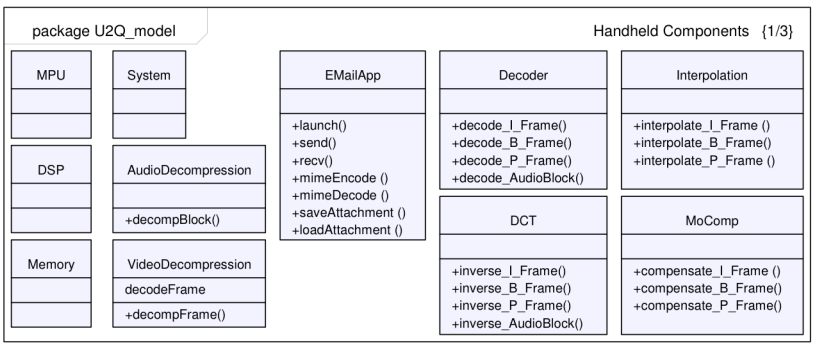
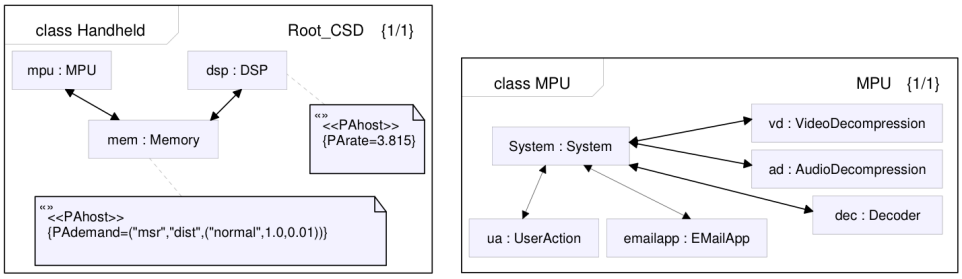



Fig. 1. Class diagram of handheld device components

In (4; 5; 8) deployment diagrams are used to model the mapping of software components to hardware. The authors of (10) suggest to use the component diagram for topology modelling. (4; 5; 10) rely on naming conventions to link behavioural entities with processing resources. (8) directly integrates the processing resource assignment into behavioural diagrams by exploiting the SPT Profile stereotype `PAhost`. In contrast to these approaches, we employ the composite structure diagram, because it allows us to semantically reuse classes as instances as its elements. In UML terminology, the elements (components) of a composite structure diagram are interpreted as objects which are instances of the specified classes. Thus, the class and interface definitions modelled as mentioned above may be reused. Components are interconnected by links and support the usage of the specified interfaces. Of course, multiple instances of the same class may be interconnected.This has the advantage that reassigning a component to another processing resource requires only changes at one place in the model by moving the component from one composite structure diagram to another. It is not necessary to modify the behavioural diagram and thus, this fulfils requirement (3). Another advantage of the composite structure diagram is its support for hierarchies of diagrams (see below). Classes can be decomposed into their internal parts by specifying the internal structure with another composite structure diagram. In this way, a hierarchy of composite structure diagram evolves. This approach can be used to iteratively refine the system model towards a sophisticated model.

Figures 2(a) and 2(b) show the internal structure of the handheld device and the `MPU` component. A data exchange path between the `MPU` and `DSP` classes in figure 2(a) is realised via the `Memory` component that enables both components to communicate. Note that the `Memory` component represents also the internal AMBA bus and thus, is also specified as an active component. In order to analyse the impact of distributing the MPEG decoding parts to either the MPU or DSP, it is only necessary to refine the `MPU` and `DSP` classes by refinement composite structure diagrams. In figure 2(b) only the `Decoder` is assigned to the MPU. A similar diagram would be used to assign the other software components to the DSP.

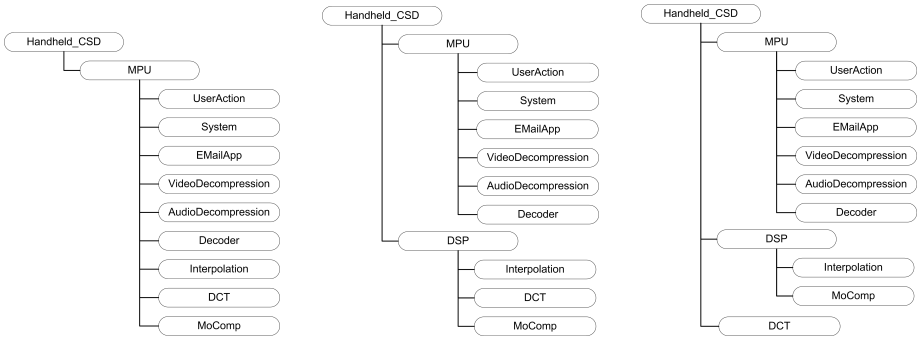



(a) Architecture of the device (b) Refinement of the MPU class

Fig. 2. Composite structure diagrams of the handheld and the MPU

Figure 3(b) presents a tree showing the composite structure diagram hierarchy for the architecture described above. We call the top level diagram (`Handheld_CSD` in

this case) the *root composite structure diagram* (root CSD). From the root CSD all other composite structure diagram are reachable in the hierarchy.




(a) CSD hierarchy with MPU only

(b) CSD hierarchy with MPU and DSP

(c) CSD hierarchy with MPU, DSP, DCT

Fig. 3. Composite structure diagram hierarchies

This method of describing the system topology does not distinguish between hardware and software, but consists only of processing resources (cf. sec. 3.3). During early stages of development it is not clear which part of the system is going to be implemented in hardware and which in software. This method allows the developers to postpone these decisions to a later design phase. In this way, the software designers only need to decide of which components their software architecture will be composed of. After evaluating different architectures, the hardware and software designers opt for a specific design, e.g., the DCT might be either implemented in software for the MPU or DSP, or even be integrated as a special purpose chip. With our methodology, these three design alternatives can be represented by just moving the `DCT` class from the MPU to the DSP or to the root CSD (cf. sec. 3; requirement (3)). The resulting composite structure diagram hierarchies for these three design alternatives can be seen in figures 3(a), 3(b), 3(c).

The final part of the system model, the dynamic behaviour, is modelled using sequence diagrams. This is a well-established approach for modelling the message exchange between different entities in a system similar to (4; 5; 2; 11; 10) which use either generic message sequence charts or also UML sequence diagrams. Again, our methodology reuses the class definitions of the system components as class instances exchanging messages in this diagram type. The messages are directly related to public methods of the particular classes (cf. sec. 3.1). Note that objects exchanging messages do not need to be directly connected in the system topology, but may communicate through intermediate components as well (cf. fig. 2(a)). In this case, it must be ensured that a communication path exists in the system topology and routing information must be gathered in order to establish a path between the sender and the receiver. This is discussed in more detail in sec. 4.1.

The sequence diagram of UML 2.0 incorporates *interaction frames* which may be used to model complex control flows. These interaction frames include `alt`-frames

(alternative execution), `loop`-frames (repeating execution), `ref`-frames (inclusion of other sequence diagrams) et al. The `par`-frame defining parallel execution with forks or joins, is excluded in our methodology, because there is no equivalent representation in queueing networks. (10) suggests to use so called “Extended Queueing Networks” to incorporate parallelism. This approach could be easily integrated into our simulative evaluation (cf. sec. 4.3). However, a valid way to model parallelism is to use several use cases in a performance scenario (cf. sec. 3.2 and 3.3) resulting in multiple workloads each with its own jobs.

Figure 4 presents a simplified sequence diagram corresponding to the video decoding process. Depending on the frame type (cf. sec. 2) the corresponding actions are run with different performance demands (cf. sec. 3.2). A similar diagram may be drawn for audio decoding.

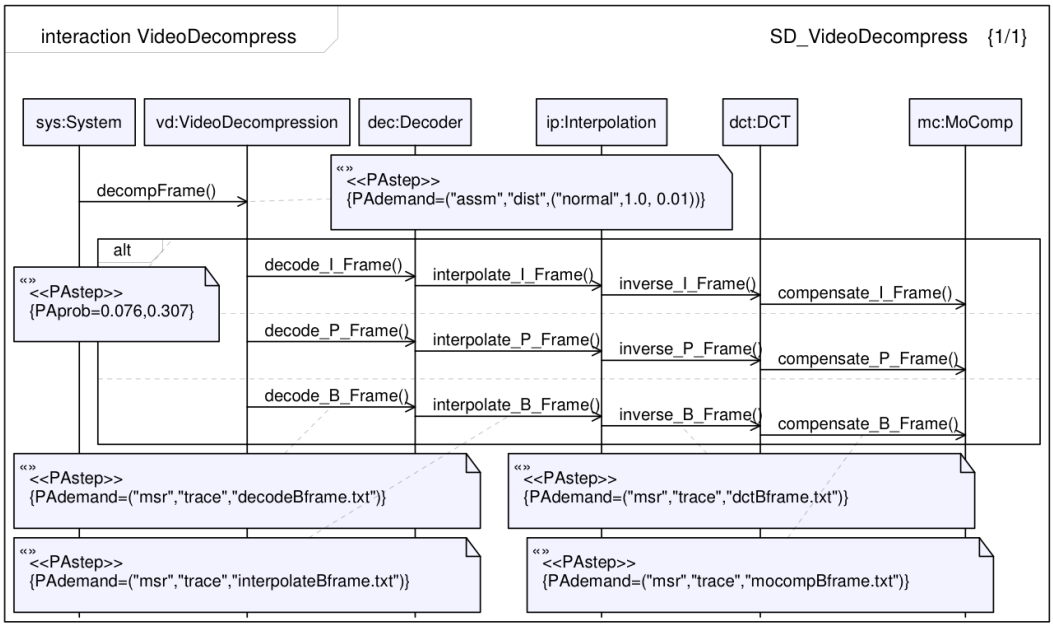



Fig. 4. Sequence Diagram of Video Decompression – this diagram contains only a subset of the necessary annotations

### *3.2 Performance Annotations*

This section shows how to incorporate performance aspects, namely performance demands and performance capacities, into the system model in a non-invasive manner such that the system model remains usable without the performance information (requirement (1)). As widely proposed, we use the ”UML Profile for Schedulability, Performance, and Time” (SPT Profile) (21) 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 modified to fit our needs which is also done by several authors e.g., (22; 10).

The beta version of the new profile "Modeling and Analysis of Real-Time and Embedded Systems" (MARTE) (23) is meant to replace the SPT Profile and is ought to fully support UML 2.0. Among other topics it focuses on the modelling of software and hardware aspects as mentioned in (24).

Two steps 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.

### *3.2.1 Component Capacity*

Similar to the system topology, the specification of the capacity of each component is an important and at the same time difficult part of the design decisions. In order to evaluate different topologies, the composite structure diagram was selected for topology modelling, because of its flexibility and the ability to easily exchange and compare it with other topologies. Thus, it is sensible to define the capacity of components in the corresponding composite structure diagram as well. In this way, the configuration of the components is as exchangeable as the topology itself. Therefore, a designer is able to exchange entire parts of the architecture at once. The alternative approach to incorporate the performance capacity into class diagrams would require to change the model at two different places in order to alter the architecture, i.e., the architecture in composite structure diagrams and the capacity in class diagrams. To model the component capacities, the SPT Profile tagged value `PArate` is used as part of the `<<PAhost>>` annotation as a relative speed value. `PArate=1` is taken as default if no annotation is provided. These values specify the relative speed factor between all components in the same diagram or hierarchy level respectively. Additionally, the scheduling policy may be specified with the `PAschdPolicy` tagged value and is set to FIFO by default. Note that using the composite structure diagram for capacity modelling violates requirement (3) as described above (cf. sec. 3.1) and is also not supported by the SPT Profile. However, the benefit of a simple architecture modification outweighs the violation and it seems to be reasonable to allow for this annotation in the upcoming MARTE Profile.

Figure 2(a) shows the application of this annotation where the component `DSP` gains a relative speed-up of 2 compared to the two other components `MPU` and `Memory`.

When modifying the relative speed of the `MPU`, it is sensible that as a consequence the refining components (e.g., the `DCT`) should 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 must be multiplied along their path through the composite structure diagram hierarchy.

The suggested application of the `PArate` tag might lead to ambivalent design de-

cisions. In general, a relative speed-up applies to every software part that is moved to a component with a higher PArate. However, only a special computing intensive kind of algorithms may benefit from the DSP. A designer who is unaware of this fact might be tempted to move every part of the software system to the DSP including even the operating system. Therefore, it is necessary to be able to restrict the annotated speed-up to stereotypes only, e.g., the DSP might be annotated with PArate=1/10 in general and with PArate=2 for the stereotyped decoding parts of the software. Only classes carrying the specific stereotype would then experience a speed-up while others would even be slowed down. Unfortunately, this is not supported by the SPT Profile, but might be an enhancement of future successors. We are planning to allow for this stereotyped speed-up in future versions of our methodology.

### *3.2.2 Resource Demand*

During the second step of incorporating the performance annotations into the model, 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. The demand is represented by a triple consisting of a source modifier specifying how the demand was captured (e.g., measured or assumed), a type modifier giving the type of the value (e.g., average value or distribution), and a time value which is the actual service time and can also be expressed by a probability distribution (for further details refer to the SPT Profile(21)).

In order to use measured data directly, we introduce the non SPT Profile conform type modifier trace which takes a filename as value. In this way, real world measured service times can be fed into the performance evaluation without the need to transform them into distributions first. During the evaluation, the service times are then adapted according to the PArate values calculated along the composite structure diagram hierarchy (cf. sec. 4.3). In this way, the developers can study how the real world service times change when they modify the system. Nevertheless, the transformation to distributions might still be sensible depending on other performance information and necessary for an analytical evaluation.

Figure 4 depicts this annotation. Please note that this figure only contains the necessary annotations for the decompression of B-frames for convenience. Both, stochastic distributions (decompFrame) as well as trace based (decode_B_Frame et al.), are shown.

We suggest to use the PAdemand tagged value also for objects in the composite structure diagram, in order to specify a default demand. In this way, the designers do not need to specify the demand of each single message exchange which leads to more convenient modelling, e.g., the latency to access memory is often the same and thus, does not need to be specified multiple times. Further, this default demand

is used, if messages are exchanged via intermediate components (see above).

### *3.3 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 aspects.

A *performance scenario* is a set of use cases and corresponding architectural system parts. Each use case is represented by a sequence diagram. The starting point of a single use case is its *root step* which is the first annotated message. It does not necessarily need to be the first message specified, but depends on the designer's interest. Architectural system parts are described by composite structure diagrams hosting the participating components (cf. sec. 3). Additionally, the performance scenario contains the workload definition for each use case. These workloads are defined by the `<<PAopenLoad>>` or `<<PAclosedLoad>>` annotations as suggested by the SPT Profile. Since these two workload annotations only specify stochastic distributions, the users are forced to transform measurements to distributions first, before they can incorporate them into their models. This has two drawbacks. First, the way how the distributions have been derived is lost and second, this makes the model harder to understand sometimes.

Therefore, we introduce a non SPT Profile conform workload definition called `<<PAtraceLoad>>` which allows us to specify a trace file similar to the trace based service times (cf. sec. 3.2.2). This workload 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. In conjunction with trace based service times, trace based workload allows for an *on-the-fly* evaluation of real world data. In this way, real time measurements of an existing system can be incorporated into the design of the not yet existing successor system.

Figure 5 shows a use case diagram with two use cases (`AudioDecompression`, `VideoDecompression`) using both types of workloads. Each use case is described by a sequence diagram, e.g., see figure 4 for the `VideoDecompression` use case.

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) of a performance scenario is the composite structure diagram which, together with its refinement CSDs, describes the topology of the components participating in the selected use cases. Thus, it is possible to evaluate only a part of the whole system by selecting a subset of the use cases and a processing CSD representing the first hierarchical layer of the topology of

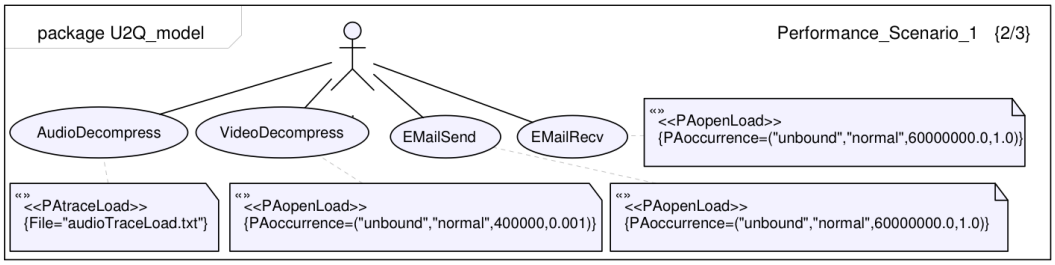



Fig. 5. Use case diagram of the performance scenario for audio and video decompression

the system part. The processing composite structure diagram is the root node of this sub-tree to which all components are mapped during the queueing network generation (cf. sec. 4). At this stage of development, we do not distinguish between hardware and software (cf. sec. 3.1), but the designers choose which components should process and execute the use cases.

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 (cf. fig. 2(b) and fig. 3) as the corresponding processing CSD without the need to specify this mapping in a new diagram.

## 4 Transformation to Queueing Networks

This section presents the algorithm that combines the system model and performance annotations 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 analytically or by simulation depending on its characteristics. If the queueing network is in product form and thus, fulfils the BCMP rules(25), an evaluation with the incorporated analytical solver is possible. Since we want to use captured inter-arrival and service times (cf. sec. 3.2.2), a simulative evaluation must be employed. Therefore, we focus on the transformation for simulations in this article only. In (26), we published a detailed description of the transformation algorithm for analytical evaluation.

A queueing network consists of interconnected queueing centers which process incoming jobs from one or several workloads. The processing is characterised by a workload dependent, stochastically distributed service time. The interconnection may include branches and feedbacks forming a loop. In order to distinguish between different visits of a job to the same queueing center, multiclass queueing networks assign a job class to each job (27).

The transformation algorithm consists of four steps as depicted in fig. 6. For each use case specified in the performance scenario, the *chain of actions* is identified which basically describes the flow of execution triggered by the reception of mes-

sages in the sequence diagrams. Thereafter, each action is *mapped to the component* performing the action by evaluating the composite structure diagram hierarchy. This information is used to generate the *queueing network* for this specific use case. Steps 1–3 are repeated for each use case until finally, in step 4, all individual queueing networks are *merged to the resulting queueing network* covering the whole performance scenario.

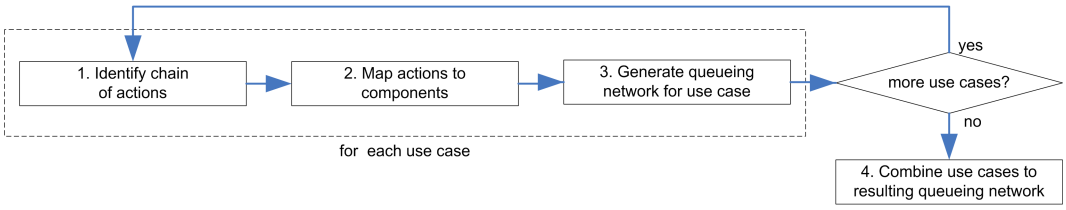



Fig. 6. The four steps of the transformation algorithm

### *4.1 Identifying the Chain of Actions*

In the first step each sequence diagram associated with the involved use cases is transformed into what we call a *chain of actions* (cf. listing 1 for a pseudocode representation described in more details below).

```
1  function IdentifyChainOfActions(SequenceDiagram or InteractionFrame u)
2    List ChainOfActions
3    foreach element in u do
4      case Message:
5        if not directlyConnected(element.source,element.destination) do
6          foreach action in findRoute(element.source,element.destination) do
7            action.demand = PAdefaultDemand(action.CSDcomponent)
8            add(ChainOfActions,action)
9          done
10       done
11       action = new Action()
12       action.demand = element.PAdemand
13       add(ChainOfActions,action)
14     case InteractionFrame:
15       case Alt:
16         add(ChainOfActions,new Alt(IdentifyChainOfActions(element)))
17       case Loop:
18         add(ChainOfActions,new Loop(IdentifyChainOfActions(element)))
19   done
20   return ChainOfActions
21 done
```

Listing 1. Identifying chain of actions

Every message in a sequence diagram corresponds to an action in the receiving object triggered by the reception of that message. Thus, the algorithm sequentially walks through the sequence diagram and creates an action for every message reception. Additionally, for each message the resource demand is identified according to its `<<PAStep>>` annotation (cf. sec. 3.2.2; lines 11-13). Loops and alternatives are

independent elements in the chain of actions and are processed individually (lines 14-18). They are resolved in a later step (cf. 4.3).

As mentioned in sec. 3.1, our methodology supports both *direct* and *indirect* message exchange. In case of an indirect message exchange, i.e., when the communicating components do not share a direct link in the system topology, a route from the source of the message to its destination component must be determined, because resources are consumed on these components as well. Note that the designer must ensure the existence of such a route. In general, such a route may be discovered using common routing algorithms such as Dijkstra, Bellman-Ford, or a simple breadth first search. The components of the composite structure diagrams correspond to nodes inside a graph for which the links are defined by the connections between the components. A drawback of this approach is that the selected route might not necessarily be the desired one if several alternative routes exist (even if it is optimal under a specific metric). To leverage this situation, additional information could be incorporated into the model to guide the algorithm in discovering the desired route, e.g., by specifying required and implemented interfaces in intermediate components. However, this additional information would violate the non-invasive principle since it modifies the system model (requirement (1)). Therefore, we apply the graph approach.

If a message is exchanged indirectly, a route is determined and the implicit actions on the intermediate components are additionally inserted into the chain of actions in front of the message target. Since these intermediate components are not annotated with a use case specific demand in the currently processed sequence diagram, the default resource demand as defined in (cf. sec. 3.2.2) is used instead (lines 5-10).

The corresponding chain of actions for the use case described by the sequence diagram in fig. 4 is depicted in fig. 7. In this example, the `Interpolation`, `DCT`, and `MoComp` are assigned to the DSP which results in the incorporation of the intermediate component *Memory* necessary for the message exchange between MPU and DSP (cf. fig. 2(a)). The default demand is taken from this CSD which is distribution ($normal(1.0, 0.01)$) in case of the *Memory* component.

### *4.2 Mapping Actions to Components*

Step 2 of the transformation algorithm takes the elements from the chain of actions and maps each of them to its executing component which may be derived from the hierarchy of composite structure diagrams (cf. listing 2).

As the resource consumption is analysed on the level of the processing CSD, all components in sub-diagrams have to be mapped to their father component in the processing CSD (cf. 3; line 26). Additionally, in order to configure the queueing network properly, the capacity of each processing resource has to be determined.

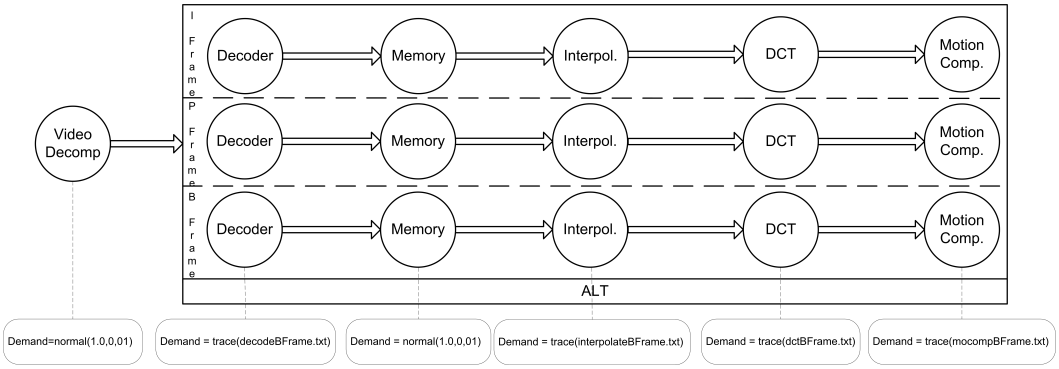



Fig. 7. Chain of actions for the sequence diagram `SD_VideoDecompress` – the resource demands are shown only for one branch

```
22 function MapComponents(ChainOfActions)
23   List Components
24   foreach action in ChainOfActions do
25     case regularAction:
26       comp = processingComponent(action.CSDcomponent))
27       add(Components,comp)
28     case Loop:
29       add(Components,MapComponents(action))
30     case Alt:
31       add(Components,MapComponents(action))
32   done
33   return Components
34 done
35
36 function processingComponent(comp)
37   if not elementOfProcessingCSD(comp) do
38     compFather = processingComponent(comp.fatherComponent)
39     compFather.capacity = comp.capacity*compFather.PArate
40     comp = compFather
41   done
42   return comp
43 done
```

Listing 2. Mapping actions to processing components

From the hierarchy of composite structure diagrams, a component's capacity may be calculated from the `PArate` tagged value as described in sec. 3.2.1, i.e., the relative speed values propagate from the processing CSD down to the refinement CSDs by multiplying the respective values. The mapping and the calculation of the capacity is determined by recursively traversing the CSD hierarchy bottom up until the processing CSD is reached. The default capacity is always 1 (lines 36-43). Loops and alternatives are still preserved and treated separately (lines 28-31).

For additional components which the processing CSD does not contain but which are required for the use case definition (e.g., source and sink in an end-to-end scenario), the corresponding components are derived from the root CSD or sub-hierarchies thereof. From the root CSD all components of the system are reachable and thus, every component can be found.

The result of this step is the sequence of processing resources in the way they are

"visited" in the current use case combined with the resource demand of the actions. Thus, each step of the sequence is annotated with the corresponding resource demand as well as the effective capacity of the associated component.

In the video decompression example, the components are mapped to the `Handheld_CSD` composite structure diagram depicted in fig. 2(a) which is used as the processing CSD. In this example, the processing CSD is also the root CSD. Figure 8 shows the modified chain of actions and the added resource capacities.

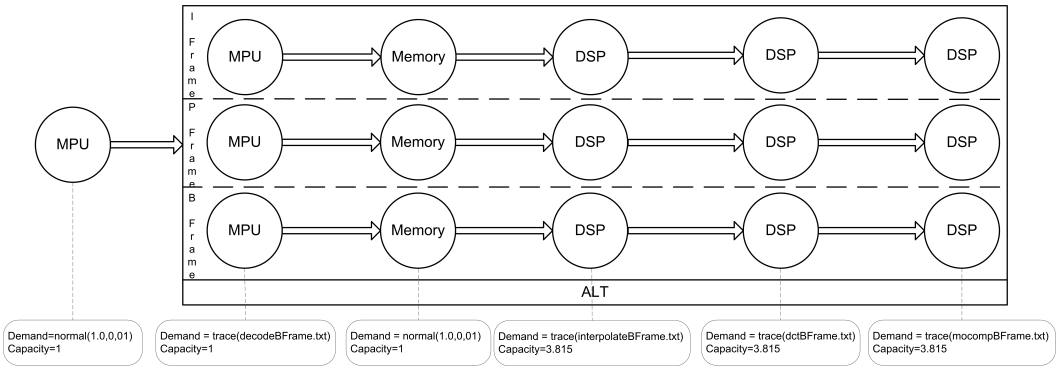



Fig. 8. Chain of actions for sequence diagram `SD_VideoDecompress` after processing component mapping – the resource demands are shown only for one branch

### *4.3 Generating a Queueing Network for a Single Use Case*

In step 3 of the transformation, the sequence of processing demands and components generated in step 2 is transformed into a queueing network corresponding to the original use case. In contrast to the analytical analysis, the sequence of the queueing centers is very important for the simulative model, because it determines the paths a job can take through the network. Thus, it is necessary to resolve the three structural elements, i.e., *explicit loops*, *implicit loops*, and alternatives in such a way that each single path through the network possibly initiated by the current workload is available in the resulting queueing network and that it is visited with the derived probability.

As mentioned above, explicit loops generated by the loop-frame essentially result in re-visiting the same queueing center several times. To be able to service each visit of a job of the same workload with a different service time, it is necessary to distinguish between each visit. For this purpose, we use the concept of multiclass queueing networks. Basically, a special job class is assigned to the job and thus, marked for looping in this way. After the job finished all iterations, the job class is changed again and the job leaves for the succeeding queueing centers. The outgoing link of the last queueing center of the explicit loop needs to be reconnected to the input of the first queueing center. In this way, the jobs of the job class to loop are

continuously rerouted back to the loop start until they finished the last repetition after which they leave the last queueing center of the loop to the remaining queueing network.

Implicit loops evolve when the same component is visited twice or more in the sequence of processing resources. This might be due to either another message sent to the same component in the sequence diagram or by mapping different refinement components to the same processing CSD component. For the implementation of implicit loops, the transformation needs to reconnect the outgoing link of a queueing center for the incoming job class to the input of the same queueing center. The job class is incremented before the connection is established to distinguish between the repeated visit. The conversion of an alternative exploits the fact that each individual branch connection to a sub-queueing network can be handled like a regular connection. Before the jobs leave the queueing center in front of an alternative, they are pseudo-randomly assigned to one of the branches according to the given probabilities. Additionally, the output of each branch is connected to the queueing center succeeding the alternative. Here, it is necessary to recombine the split job stream by assigning a uniform job class to every job leaving the different branches. Listing 3 shows a pseudocode representation of the algorithm.

```
44 HashMap QCs, List QN, List lastQC
45 function generateQN(Components)
46   List oldLastQC = lastQC
47   foreach comp in Components do
48     case Component:
49       qc = get(QCs,comp) or put(QCs,comp,makeNewQC(comp.PAschdPolicy))
50       jc = qc.nextJobClass
51       setServiceTime(qc,jc,comp.demand/comp.capacity or (comp.tracefile,comp.
             capacity))
52       foreach element in lastQC do connect(lastQC,qc,jc)
53       lastQC = qc
54     case Loop:
55       generateQN(getLoopComponents(comp))
56       lastQC = comp.lastElement
57     case Alt:
58       foreach branch in comp do generateQN(getAltComponents(branch))
59       lastQC = oldLastQC
60       foreach branch in comp do add(lastQC,branch.lastElement)
61   done
62 done
```

Listing 3. Generating use case queueing network

Each participating component is transformed into a queueing center. The service times for each queueing center are calculated by dividing the resource demand as derived in the chain of actions and the resource capacity as derived in step 2. If trace based service times are specified (cf. sec. 3.2.1), the capacity is saved and the service time is reduced during the simulation execution. The scheduling policies of the queueing centers are taken from the `PAschdPolicy` tagged value of the processing CSD `<<PAhost>>` annotations (lines 47-51). Loops and alternatives are unfolded by first, transforming them as individual queueing networks and then, connecting

these sub-queueing networks to the main queueing network (lines 54-60).

In case of an alternative, there is more than one preceding queueing center to connect to the succeeding network. In order to join the alternatives, each single branch has to be reconnected to the succeeding queueing center. By using the same job class for each connection, it is ensured that the branches will be combined to the same original job stream again (line 52).

Figure 9 depicts the queueing network for the video decompression modelled by the sequence diagram of figure 4. In this particular transformation, parts of the decompression algorithm are processed by the MPU and parts by the DSP. The decision of which part should be processed by which component is derived from the composite structure diagrams as described in section 3.1, i.e., the `VideoDecompression` algorithm and its part `Decode` run on the MPU while the `Interpolation`, `MoComp`, and `DCT` are delegated to the DSP. The `decompFrame()` method of the class `VideoDecompression` is reflected by the full arrow labelled "Video". After being processed for the first time, the jobs are separated into three different job classes due to the alternative distinguishing between the three MPEG frame types and rerouted back to the MPU (`Decode`). At last, the jobs move on via the Memory queueing center to the DSP queueing center (`Interpolation`). There they are implicitly looped for two more times (`DCT`, `MoComp`).

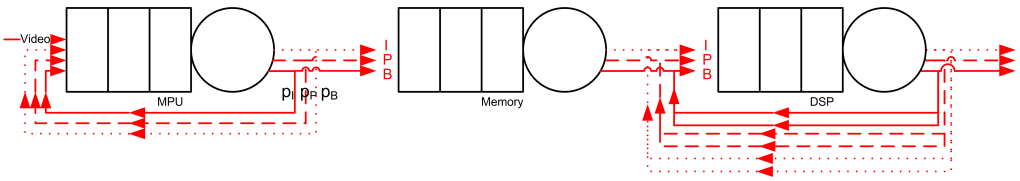



Fig. 9. Resulting queueing network for `SD_VideCompress` sequence diagram

### *4.4 Combining Use Cases to the Resulting Queueing Network*

The last transformation step merges the use case specific queueing networks of step 3 to a resulting queueing network representing the entire performance scenario.

In this step, it has to be assured that the job classes assigned in the use case specific queueing networks are mapped to unique job classes of the resulting queueing network. This allows the queueing centers to distinguish between the use cases, and thus to process jobs of different workloads with the appropriate service times. Listing 4 shows the pseudocode for this last step.

In the example performance scenario, there are two sequence diagrams leading to the resulting queueing network depicted in fig. 10. Each sequence diagram is represented by an individual workload, i.e., `Audio` for the audio decompression and `Video` for the video decompression.

```
63 function mergeQNs(QNs)
64   List resultingQN, HashMap QCs, uniqueJC = 0
65   foreach qn in QNs do
66     exchangeJC(qn,uniqueJC)
67     uniqueJC = uniqueJC + 1
68     foreach qc in qn do
69       globalQC = get(QCs,name(qc))
70       if not globalQC do
71         globalQC = qc
72         add(QCs,name(qc),qc)
73       else
74         globalQC = combine(globalQC,qc)
75       done
76     done
77   done
78   return resultingQN
79 done
```

Listing 4. Generating resulting queueing network

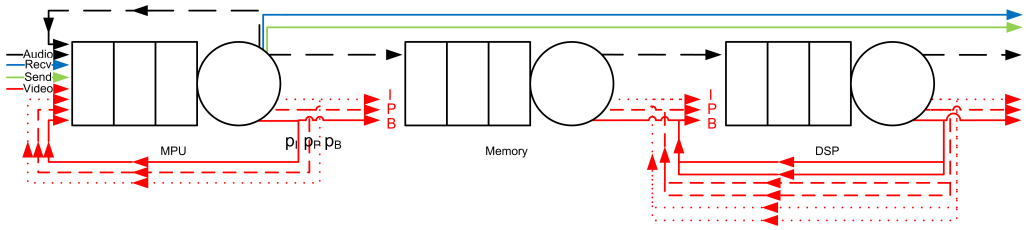



Fig. 10. Resulting queueing network for Audio, Video, and e-mail use cases

## 5 Performance Analysis

In this section, we present the performance analysis steps taken for the described scenario (cf. sec. 2). 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. See figures 3(a) and 3(b) for the corresponding composite structure diagram hierarchies. 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. Sending and receiving of e-mails was added to consider the impact of background tasks. Section 5.1 describes how we obtained the necessary performance figures for specifying the performance annotations. In section 5.2 we present the results of our simulations.

We have implemented the described methodology and integrated the resulting tool into Telelogic Tau G2(28), a UML 2.0 CASE environment, for an easy application. Our *UML to Queueing Networks* (U2Q) tool generates input for the general purpose simulator Omnet++(29) which was enhanced where necessary. The transformation process can be configured and started directly from Tau G2 using a self made add-in. This allows the developers to run the evaluation without switching to another application. This add-in is able to analyse the evaluation output and to write back

the results 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 HTML format. This seamless integration of the performance evaluation into a CASE tool has two benefits. Queueing network theory as the basis for the evaluation is hidden as much as possible from the designer. This allows the designers to use performance engineering even if they are not experts in this theory. Only by hiding the underlying mechanisms, performance engineering might be adopted by the designers. This is complemented by the fact that the designers do not need to switch applications to use the evaluation tool, but may start it directly from their well-known design application. The current version of U2Q along with the Tau G2 3.1 add-in can be downloaded at (30).

### *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 2, 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 running Linux. 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. 2). 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*(31) used by the OpenZaurus(32) 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.

| | **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

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 for convenience while the video quality was reduced. 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 log-normally distributed. Figure 11 depicts the pdf of the video frame decoding times and a fitted log-normal distribution with the parameters $\mu = 10.45, \sigma = 0.31$. The decoding times of the different frame types can be well distinguished by the three peaks in the sample pdf. Nevertheless, the log-normal 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. Table 2 presents all distribution parameters used for the stochastic evaluation.

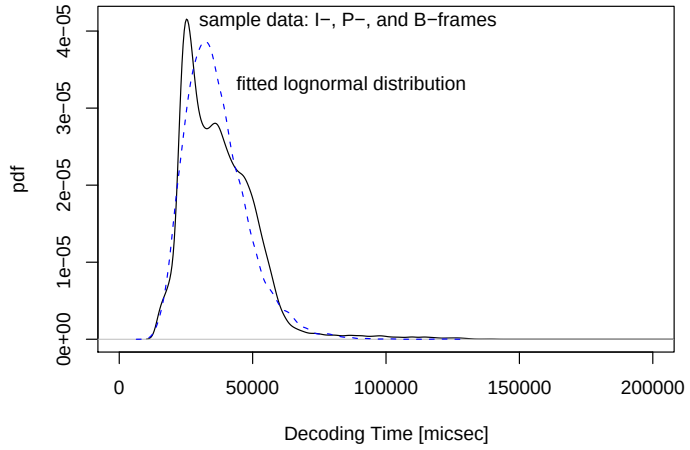



Fig. 11. Density of frame decoding times and fitted log-normal

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

Table 2
Parameters for service time modelling using log-normal 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 (33) break down the MPEG decoding algorithm into its 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 com-

plexity, interpolation 12%, and motion estimation is performed 10% of the time. In (34), 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, interpolation may be expected to execute 7.3 times faster, motion compensation 5.2 times faster, and DCT may be expected to accelerate by a factor of 4.1.

The goal of this evaluation was to inspect whether a DVB-H stream can be decoded using an OMAP platform and still having sufficient resources for other tasks like processing e-mails. We measured the MPU computing time (system and user) for receiving and sending e-mails with a 4 KB attachment on an idle system. Both tasks include MIME en-/decoding and saving/loading the attachment. First, we measured the MPU processing time (kernel and user) for receiving 1000 e-mails by an SMTP server process. While it is reasonable to assume that such a process, once started, continuously runs in the background, sending e-mails is often accompanied by starting the corresponding application first. Therefore, we measured the invocation and sending time for 1000 e-mails. Both measurement series showed to be very stable with only marginal variances. The mean reception took 0.02348 sec kernel time and 0.27438 sec user time. Sending an e-mail took 0.5345 sec kernel time and 7.669 sec user time. As a worst case scenario to stress the MPU, we assumed that one e-mail is received and send every minute.

### *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 lead to a significant reduction of computation times. According to the results of (33) 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. 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(a) shows the utilisation of the MPU and DSP in case that the motion compensation remains to be processed by the MPU. Table 3(b) presents results with all

three mentioned algorithm parts running on the DSP which accounts for 81% of the computational complexity. Finally, table 3(c) contains the results after incorporating the e-mail use cases. It can be clearly seen that both processors have enough resources to perform additional background tasks and thus, it is sensible to invest into porting parts of the decoding algorithms to the DSP.

| Sim. run | **MPU** | **DSP** |
|---|---|---|
| stoch. | 93% | 50% |
| trace | 91% | 49% |

(a) DSP runs DCT and interpolation

| Sim. run | **MPU** | **DSP** |
|---|---|---|
| stoch. | 63% | 57% |
| trace | 62% | 56% |

(b) DSP runs DCT, interpolation, and motion compensation

| Sim. run | **MPU** | **DSP** |
|---|---|---|
| stoch. | 97% | 57% |
| trace | 95% | 56% |

(c) MPU processes E_Mails

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

In addition, we ran the simulation of architecture (2) with the trace files generated by our measurements. In these trace files the decoding function, 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 Summary & Future Work

In this article we presented a performance engineering methodology and its application which mainly aims on non performance experts. It combines functional and non-functional, i.e., performance, aspects in one UML model. while not altering the functional model and keeping it working independently. In this way, non performance engineers can still use the system. From this enhanced system model, performance models were derived by an implemented transformation algorithm which was also described. These performance models were analysed with queueing network theory and provided a performance estimation of the modelled system and thus, allow for the evaluation of the system design. The seamless integration of the implementation into the CASE-tool Tau G2 hides queueing network theory almost completely from the designer and thus, simplifies the application of our methodology and of performance engineering in general for non-experts.

The presented system modelling approach proved to be applicable and well suited to analyse a multimedia enabled handheld device. The intuitive separation of aspects into components, architecture, and behavioural model is strictly kept during all stages of the system design. Especially composite structure diagrams allowed us to iteratively decompose the system in an elegant fashion.

The performance annotations describing the resource capacities and demands were modelled with UML SPT Profile annotations. Where reasonable or necessary we adapted or extended the profile annotations. Since the capacities are rather part of the architectural system design, we suggest to use the `PArate` and `PAdemand` annotations in composite structure diagrams. This allows us to specify default demands which are used if no specific demand is specified in the use case, e.g., if intermediate components have to be inserted between two components communicating indirectly. Even though this violates the required separation of component and architecture modelling, it enables an easy exchange and comparison of complete architectures. Further, we introduced the tagged annotation `PAtraceLoad` and the type modifier `trace` for the tagged value `PAdemand` in order to incorporate real world measurements directly into the model.

The integration of our U2Q tool into the UML CASE tool Tau G2 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.

Future work will concentrate on extending the methodology and on the implementation. The limitation of speed-ups modelled by the `PArate` tagged value to only a specific type of methods or classes is very important as already mentioned in the text. For this purpose stereotypes might be used. The support of activity diagrams for the dynamic behaviour modelling is an extension we are currently working on. Further, it is sensible to transit to the upcoming MARTE Profile which already incorporates solutions for the drawbacks of SPT we mentioned.

## References


[1] L. Pustina, V. Deichmann, M. Gerharz, P. Martini, S. Schwarzer, Performance evaluation of a dvb-h enabled mobile device system model, in: Proceedings of the 6th International Workshop on Software and Performance WOSP 2007, 2007, pp. 164–171.

[2] L. G. Williams, C. U. Smith, Performance evaluation of software architectures., in: WOSP, 1998, pp. 164–177.

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

[4] V. Cortellessa, R. Mirandola, Deriving a queueing network based performance model from uml diagrams., in: Workshop on Software and Performance, 2000, pp. 58–70.

[5] V. Cortellessa, M. Gentile, M. Pizzuti, Xprit: An xml-based tool to translate uml diagrams into execution graphs and queueing networks., in: QEST, 2004, pp. 342–343.

[6] S. Bernardi, S. Donatelli, J. Merseguer, From uml sequence diagrams and statecharts to analysable petri net models, in: WOSP '02, ACM Press, New York, NY, USA, 2002, pp. 35–45.

[7] J. Merseguer, J. Campos, Software performance modeling using uml and petri nets., in: M. Calzarossa, E. Gelenbe (Eds.), MASCOTS Tutorials, Vol. 2965 of Lecture Notes in Computer Science, Springer, 2003, pp. 265–289.

[8] S. Balsamo, M. Marzallo, Performance evaluation of uml system architectures with mutliclass queueing network models, in: WOSP, 2005, pp. 37–42.

[9] M. Marzolla, S. Balsamo, Uml-psi: The uml performance simulator., in: QEST, 2004, pp. 340–341.

[10] A. D. Marco, P. Inverardi, Compositional generation of software architecture performance qn models., in: WICSA, IEEE Computer Society, 2004, pp. 37–46.

[11] J. Xu, C. M. Woodside, D. C. Petriu, Performance analysis of a software design using the uml profile for schedulability,

performance, and time., in: P. Kemper, W. H. Sanders (Eds.), Computer Performance Evaluation / TOOLS, Vol. 2794 of Lecture Notes in Computer Science, Springer, 2003, pp. 291–307.
[12] D. C. Petriu, X. Wang, From uml descriptions of high-level software architectures to lqn performance models, in: AGTIVE '99: Proceedings of the International Workshop on Applications of Graph Transformations with Industrial Relevance, Springer-Verlag, London, UK, 2000, pp. 47–62.
[13] S. Balsamo, A. D. Marco, P. Inverardi, M. Simeoni, Model-based performance prediction in software development: A survey., IEEE Trans. Software Eng. 30 (5) (2004) 295–310.
[14] D.-H. org, Dvb-h homepage, http://www.dvb-h.org.
[15] ISO, Iso/iec 14496, http://www.iso.ch.
[16] T. Wiegand, G. J. Sullivan, G. Bjntegaard, A. Luthra, Overview of the h.264/avc video coding standard, Circuits and Systems for Video Technology, IEEE Transactions on 13 (7) (2003) 560–576.
[17] Omap platform, www.omap.com.
[18] V. Cortellessa, R. Mirandola, Prima-uml: a performance validation incremental methodology on early uml diagrams., Sci. Comput. Program. 44 (1) (2002) 101–129.
[19] R. Pooley, P. King, The unified modeling language and performance engineering, in: IEE Proceedings — Software., 1999, pp. 2–10.
[20] ETSI, Uml profile for communicating systems (draft) (2005).
[21] OMG, UML Profile for Schedulability, Performance, and Time Specification 1.0, Object Management Group, 2003.
[22] M. Woodside, D. C. Petriu, D. B. Petriu, H. Shen, T. Israr, J. Merseguer, Performance by unified model analysis (puma), in: WOSP'05 Proceedings of the 5th international workshop on software and performance, ACM Press, 2005, pp. 1–12.
[23] OMG, A UML Profile for MARTE, Beta 1, Object Management Group, 2007.
[24] H. Espinoza, H. Dubois, J. Medina, S. Grard, A general structure for the analysis framework of the uml martes profile (Oct 2005).
[25] F. Baskett, K. M. Chandy, R. R. Muntz, F. G. Palacios, Open, closed, and mixed networks of queues with different classes of customers., J. ACM 22 (1975) 248–260.
[26] L. Pustina, V. Deichmann, M. Gerharz, P. Martini, S. Schwarzer, Performance aware design of communication systems, in: Proceedings of LCN 2006, 2006, pp. 39–46.
[27] R. Jain, The Art of Computer Systems Performance Analysis, Wiley Professional Computing, New York, 1991.
[28] Telelogic Tau G2 homepage, `http://www.telelogic.com`.
[29] Omnet++ – general purpose network simulator, http://www.omnetpp.org.
[30] Homepage of U2Q, http://www.cs.uni-bonn.de/IV/U2Q.
[31] MPlayer, http://www.mplayerhq.hu.
[32] OpenZaurus, http://www.openzaurus.org.
[33] K. Ramkishor, V. Gunashree, Real Time Implementation of MPEG-4 Video Decoder on ARM7TDMI, in: Intelligent Multimedia, Video and Speech Processing, 2001, pp. 522–526.
[34] J. Chaoui, K. Cyr, S. de Gregorio, J.-P. Giacalone, J. Webb, Y. Masse, Open multimedia application platform: enabling multimedia applications in third generation wireless terminals through a combined risc/dsp architecture, in: ICASSP '01: Proceedings of the Acoustics, Speech, and Signal Processing, 200. on IEEE International Conference, IEEE Computer Society, Washington, DC, USA, 2001, pp. 1009–1012.