technische universiteit eindhoven

An Overview of Application Scenario Usage in Streaming-Oriented Embedded System Design

Stefan Valentin Gheorghita, Twan Basten, Henk Corporaal

ES Reports
ISSN 1574-9517

ESR-2006-03
20 May 2006

Eindhoven University of Technology
Department of Electrical Engineering
Electronic Systems

Get the habit of WCET analysis - it will, in time, enable synthesis for real-time to become your habit of mind.

http://www.es.ele.tue.nl/esreports
esreports@es.ele.tue.nl

Eindhoven University of Technology
Department of Electrical Engineering
Electronic Systems
PO Box 513
NL-5600 MB Eindhoven
The Netherlands

An Overview of Application Scenario Usage in Streaming-Oriented Embedded System Design*

Stefan Valentin Gheorghita, Twan Basten and Henk Corporaal
EE Department, Electronic Systems Group
Eindhoven University of Technology, PO Box 513,
5600MB, Eindhoven, The Netherlands
{s.v.gheorghita,a.a.basten,h.corporaal}@tue.nl

20 May 2006

Abstract

In the past years real-time embedded systems became more and more complex. From the user perspective, these systems have stringent requirements regarding size, performance and power consumption, and due to business competition, their time-to-market is a crucial factor. Therefore, much work has been done in developing design methodologies for embedded systems to cope with these tight requirements. In this report, we present a basic methodology based on the concept of *application scenarios* for real-time embedded systems design. We show the distinction between application scenarios and use-case scenarios. Moreover, we give a literature overview of application scenario usage in embedded system design for obtaining a faster implementation, an energy saving implementation, or a better estimation of the resources required by an application.

Keywords: Real-Time Embedded Systems, Application Scenarios.

1 Introduction

Embedded systems usually consist of processors that execute domain-specific programs. These systems are software intensive[1], having much of their functionality implemented in software, which is running on one or multiple generic processors, leaving only the high performance functions implemented in hardware. Typical examples of embedded systems include TV sets, cellular phones, MP3 players and printers. Most of these systems are running multimedia and/or telecom applications, like video and audio decoders. These applications are usually implemented as a main loop, called the loop of interest, that is

[*] This work was supported by the Dutch Science Foundation, NWO, project FAME, number 612.064.101.

[1] If the system's software contributes with essential elements to the design, construction, deployment, and evolution of the system as a whole, we talk about a software intensive system [IEE00].

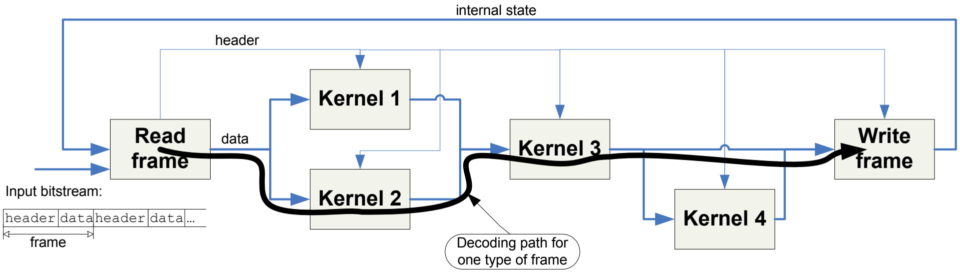

Figure 1: Typical multimedia application decoding a frame.

executed over and over again, reading, processing and writing out individual stream objects (see figure 1). A stream object might be a bit belonging to a compressed bitstream representing a coded video clip, a macro-block, a video frame, or an audio sample. Usually, these applications have to deliver a given throughput (number of objects per second), which imposes a time constraint on each loop iteration. For the sake of simplicity, and without loss of generality, from now on we use the word frame to refer to a stream object.

The read part of the loop of interest takes a frame from the input stream and separates it in a *header* and the frame's *data*. The processing part consists of several kernels. For each frame some of these kernels are used, depending on the frame type. The write part sends the processed data to the output devices, like a screen or speakers, and saves the internal state of the of the application for further use (e.g. in a video decoder, the previous decoded frame may be necessary for decoding the current frame). The actions executed in a loop iteration form an internal *operation mode* of the application.

In this work, we introduce ways of detecting and exploiting a characteristic of the application that has not been fully used in embedded systems design previously, namely that many applications have different internal operation modes, each with their own typical resource consumption. We suggest to cluster operation modes which are closely related to each other from a resource usage point of view in so-called application scenarios, distinguishing operation modes that are really different[2]. If these scenarios are considered in different steps on the embedded systems design, a faster implementation (e.g. by using different source code optimizations per scenario), an energy saving implementation, or a better estimation of the resources required by an application (e.g. the number of computation cycles or bandwidth) may be derived. These intermediate results lead to a *smaller, cheaper and energy efficient system that can deliver the required performance*.

The report is organized as follows. Section 2 emphasizes the differences between application scenarios and the well known use-case scenarios. A definition for application scenarios is given in section 3. Their classification together with a basic methodology of incorporating them in an embedded system design flow is presented in section 4. A literature overview and few conclusions about using application scenarios in embedded system design are presented in the last two sections.

2 Scenarios in Design

Scenario based design has been used for a long time in different areas [Car95], like human-computer interaction [RC02] or object oriented software engineering [Jac95]. In both these cases, scenarios concretely describe, in an early phase of the development process, the use of a future system. In case of human-computer interaction, the scenarios appear like narrative descriptions of envisioned usage episodes, and in case of object oriented software engineering like a unified modeling language (UML) use-case diagram [Fow03] which enumerate, from functional and timing point of view, all possible user actions and the system reactions that are required to meet a proposed system function. These scenarios are called *use-case scenarios*.

In the embedded systems area, use-case scenarios were used in both hardware [Ion05], [Pau05] and software design [Dou04]. In these cases, the scenarios focus on the application functional and timing behaviors and on its interaction with the users and environment, not on the resources required by an application to meet its constraints. Moreover, the scenarios are described by the designer in a narrative or graphical way. These scenarios are used as an input during application design for user-centered design approaches.

In this work, we concentrate on a different type of scenarios, so called *application scenarios*, which may be semi-automatically derived from the internal behavior of the application with respect to the amount of hardware resources required by the application to meet its constraints. These scenarios pro-

[2] The clustering decision is a complex process as it has to take into account trade-offs in a multi-dimensional objectives (e.g. resources) design space.

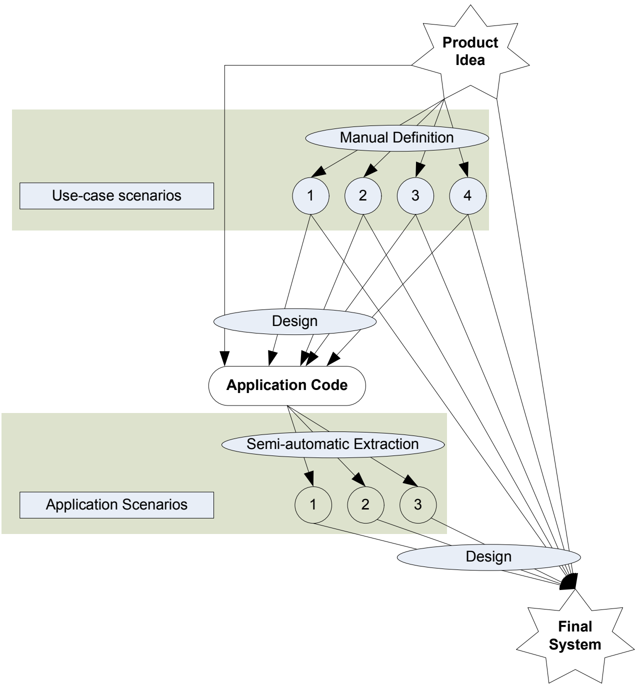

Figure 2: A scenario based design flow for embedded systems.

vide internal application information and they are used to improve the system design, to create cheap embedded systems in which the application uses the hardware resources efficiently.

Figure 2 depicts a design trajectory using scenarios. It starts from a product idea, for which the stakeholders[3] define the product future utilization as use-case scenarios. These scenarios characterize the system from a *user perspective* and are used in a user-centric development process to design an embedded system that includes both software and hardware components. In order to optimize the design of the system, the detection and usage of application scenarios augments this trajectory (the bottom gray box from figure 2). Once the application is developed, in a semi-automatic way, its scenarios related to the system resource utilization are extracted, and they are considered in the decisions made during the following phases of the system design. The application scenarios characterize the system from the *resource usage perspective*. The sets of use-case scenarios and application scenarios are not necessarily disjoint, and it is possible that one or more use-case scenarios are merged in one application scenario, a use-case scenario is split into several application scenarios, or that several application scenarios intersect several use-case scenarios.

Example: Let us consider that we want to design a small portable MP3 player as a USB stick. At first sight, there are two main use-case scenarios: (i) the player is connected to the computer and music files are transferred between them, and (ii) the player is used to play music. These scenarios can be divided in more detailed use-case scenarios, like, for the second one, we may have the song selection, playing or fast forward scenarios.

Let us consider the playing use-case scenario. From the software point of view, this use-case can be split in two different application scenarios: playing songs (i) in mono mode and (ii) in stereo mode. If these scenarios are detected in the application, and the information about them is used during the system design, the system battery lifetime may be increased, as in case of playing in mono mode a lower computation power is needed, thus a lower supply voltage may be used to meet the timing constraints of the decoding.

It can be observed that in this scenario hierarchy, there can be multiple levels with the same type

[3] The stakeholders are persons, entities, or organizations who have a direct stake in the final system; they can be owners, regulators, developers, users or maintainers of the system.

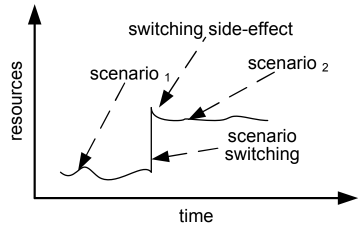

Figure 3: Example of an application execution histogram

of scenarios, each level being just a refinement. At the boundary between use-case and application scenarios, it is not necessary a true hierarchy, since application scenarios may intersect with use-case scenarios in some arbitrary way, as mentioned above.

3 Definition(s)

The most general definition for a scenario, which covers both use-case and application scenarios, is: *a scenario is a certain mode in which an application operates.* As already presented, a use-case scenario is a story that describes a (future) use of a system. A survey of different definitions found in the literature for use-case scenarios is presented in [Ion05]. Our definition of application scenarios is as follows.

Definition (APPLICATION SCENARIO): *An application scenario is a runtime detectable set of operation modes of an application that are sufficiently similar in an N-dimensional cost space (e.g. speed, memory usage, source code).*

The cost space is defined over the dimensions of interest for a specific problem. For example, we might be interested in scenarios which share the same source code, or in scenarios which use the same amount of resources (e.g. CPU cycles, memory size, or communication bandwidth). In order to be exploited, these modes of operation must be detectable at runtime, preferably as soon as the application starts to execute in them. The set of selected scenarios must cover the entire application, and should lead to a global minima for the problem in hand.

As the previous definition is a very general one, it is commonly tailored for the specific design problem at hand. Few examples are:

- a *part* of the application source code with a specified maximum number of loop iterations ([GSBC05b]);
- the application behavior for a specific type of input data ([GBC05]);
- a group of execution paths for a particular group of input data ([PCC05], [PBV$^+$06]).

Several papers presented in the literature overview from section 5 use scenarios without giving an explicit definition and/or identifying the concept.

Definition (RUNTIME SCENARIO SWITCHING): *The activity done at runtime when the application switches from one application scenario to another application scenario is called runtime scenario switching.*

The switching between scenarios adds at runtime an additional cost in term of used resources.

Example: Figure 3 shows an example of an application execution histogram, displaying on the horizontal axis the passing of time and on the vertical axis the amount of used resources. It can be observed that the operation modes that are similar from a resource consumption point of view are combined together in the same scenario. A scenario switch may cause some short peak in resource usage.

For the sake of simplicity, in the remainder, we use the term scenario to refer to an application scenario.

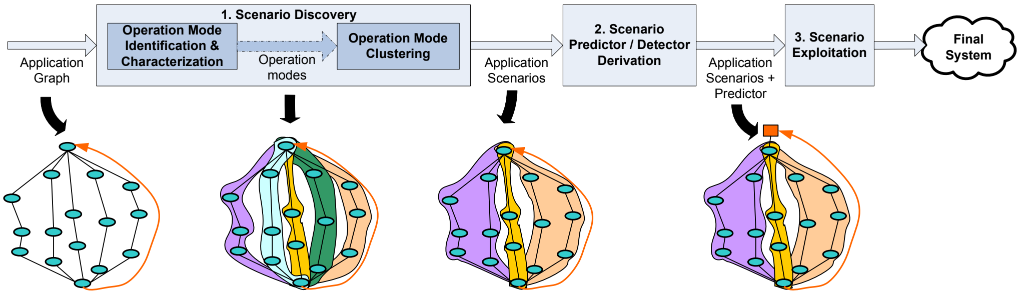
1. Scenario Discovery
Operation Mode Identification & Characterization
Operation modes
Operation Mode Clustering
Application Graph
Application Scenarios
2. Scenario Predictor / Detector Derivation
Application Scenarios + Predictor
3. Scenario Exploitation
Final System

Figure 4: Application scenario usage methodology

4 Methodology & Classification

In this section, a basic methodology of using application scenarios in an embedded system design trajectory is detailed. Furthermore, a scenario classification is made based on different properties/types of embedded systems.

4.1 Methodology

The methodology of using application scenarios in embedded system design may be divided in three steps that are depicted in figure 4:

1. *Scenario identification or discovery* starts from the original application and identifies the different operation modes that appear in it. The modes with similar needs of resources (e.g. processing time, data memory, instruction memory) are clustered together and form the application scenarios. Operation modes identification and the clustering of modes can be done in sequence or simultaneously. The first case allows in theory a better exploration of the design space, and the second one limits its size explosion. Depending on the number of resources of interest (N), the clustering problem may become very complicated, especially because the exact definition of similarity based on a small enough distance between two operation modes in an N-dimensional space is very ambiguous.

2. The runtime scenario *detector* and/or *predictor* derivation is the step of finding a way to detect or predict in which scenario the application runs at a certain moment in time, in order to be able to exploit the scenarios during runtime. The current scenario in an application can be either *detected*, based on already known internal information of the application (e.g. variable values), or it can be *predicted*, with a certain confidence, based on internal information, scenario history and statistical information regarding runtime switching between scenarios. We can say that detection is prediction with 100% confidence.

3. *Scenario exploitation* is the step which uses the information provided by the scenarios in order to optimize the design problem that we want to solve. An example of exploitation could be the generation of the application implementation optimized for energy saving. This may involve different optimization criteria, like minimizing application code size and scenario switching time overhead. This step depends on and should take into account the intended usage.

Besides the previous three steps which are done during design time, there could be an additional one, so called *scenario activation*, that during runtime, based on the information provided by the scenario predictor, startups the appropriate program code related to the scenario and switch to other scenario when appropriate. The source code to provide the actions for this step are generated and introduced into the application during design time, especially in steps 2 and 3.

The following two sections discuss the steps 1 and 2; the scenario exploitation step is not detailed, as it strongly depends on the problem to be solved. Different problems are presented in the literature overview presented in section 5.

4.2 Scenario identification/discovery

The methods for application scenario identification can be divided in three categories: analytical, profiling and hybrid. Independent of the used method, the set of the identified application scenarios must cover all possible application operation modes.

- *Analytical*: The application structure is statically analyzed in different ways in order to identify similar situations in the application. This method is restrictive, as it can not automatically collect information about how the application is really used and how it behaves at runtime (e.g. which is the most frequently used scenario at runtime). This information can still be provided by the designer.

- *Profiling*: The real runtime behavior of the application can be captured, but it is more difficult to derive the scenario predictor/detector than it is in the analytical case. This appears due to the fact that a purely profiling approach does not take into account any analytical discovered knowledge about the application source code. By using this method, it is almost impossible to detect all scenarios as not all possible distinct application operation modes may be covered by profiling. Because of this, an extra scenario, called the *backup scenario* must be considered. At runtime, this scenario is selected when the application is running in an operation mode that was not covered during the profiling used to identify its scenarios.

- *Hybrid*: This method combines the previous two methods and it is the most powerful way to identify scenarios, especially for the case when the scenarios are used at runtime.

Especially for profiling and hybrid methods, an explosion in the number of scenarios may appear during their identification. In this case, decisions must be made using partial information, applying the scenario discovery and clustering simultaneously. This means that there is a trade-off between how many different scenarios may be handled during the identification process and how accurate the identification will be. The optimal number of scenarios depends on many factors, including their intended usage.

Besides the bottom-up approach that first identifies a large set of scenarios in the application, and afterwards, tries to combine them based on their similarity, there is also a top-down approach. This one considers first the application as a single scenario, and then, recursively splits each scenario based on the differences between the operation modes covered by it. The trade-off due to the explosion problem could also appear in this approach: either all the modes are first analyzed and then the application is split into scenarios, or the decision is made based on intermediate information. The two approaches are not fundamentally different, being just different ways to implement the scenario discovery problem.

4.3 Scenarios prediction/detection

Considering that a set of scenarios was identified in the application, ways of detecting the currently running scenario and switching between scenarios at runtime are needed. The former can be implemented as a *predictor* or as a *detector*. The difference between them is that a *detector* can say with 100% confidence in which scenario the application is running, whereas a predictor can just suppose it with lower confidence. In the remaining part of this section, the verb *predict* is used to commonly characterize the operations done by both predictors and detectors.

From a structural point of view, both the predictors and the detectors can be classified as follows:

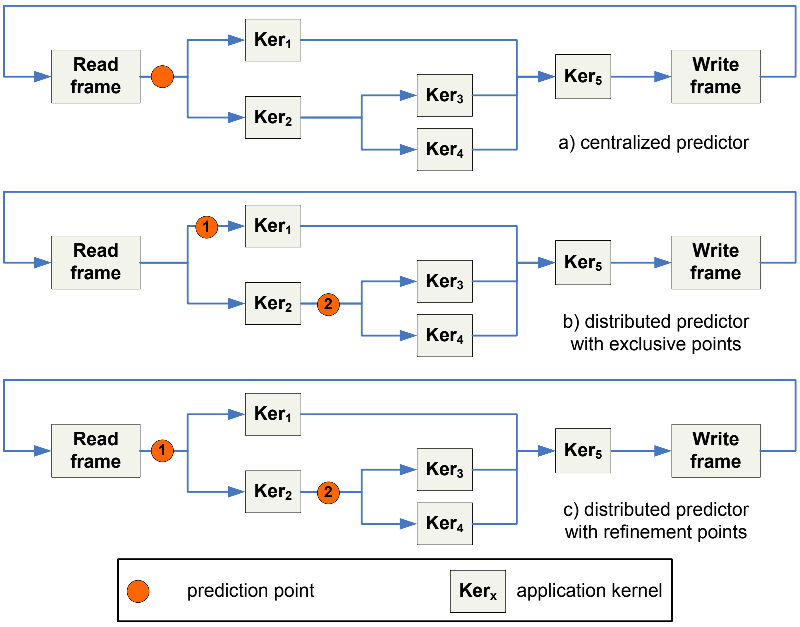

Figure 5: Different types of predictors from the structural point of view

- *Centralized*: There is only one central point in the application where the current scenario is predicted. It is inserted in the application code in a common place that appears in all scenarios (e.g. for the application model presented in figure 5 (a), it is introduced in the main loop, after the read part, when all the information necessary to predict the current scenario is known).

- *Distributed*: There are multiple points for predicting the application scenarios, which may be:

  - *Exclusive points*: Only one point from the set is executed in each scenario prediction. Figure 5 (b) depicts a case where one of two prediction points is being executed for different iterations. The used predictors may or may not be different.

  - *Refinement points*: Multiple points are used to predict the current scenario; the first one will predict a set of possible scenarios and possibly their probabilities, and the following will refine the set until only one scenario will remain. For the application depicted in figure 5 (c), considering the scenario that executes kernels two, three and five, the first prediction point can predict a set of scenarios, and the second prediction point is able to predict the right scenario.

When implementing predictors and detectors the following information may be considered[4]:

- runtime application internal information like variable values, executed code (i.e. a basic block that appears only in one scenario);

- statistical information, often obtained by profiling or from the application designer, about how often a scenario may appear at runtime;

- probabilistic scenario transition model, like a Markov chain model, that may be obtained by application profiling (this includes the information from the previous step);

- history of active scenarios in the current application run;

- knowledge about the correlation that appear within the input streams.

[4] In reality, some of the information can not be used for detectors as they will not have a 100% confidence.

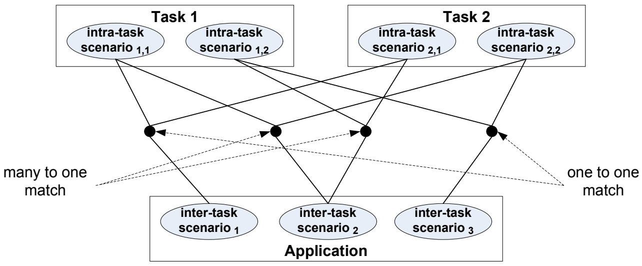

Figure 6: Possible relations between intra- and inter-task scenarios

Considering the way how both predictors and detectors use application internal information, they may be classified as:

- *Reactive*: The only internal information from the application that they use in the prediction is the one already computed by the application.
- *Proactive*: A part of the application control-flow that follows the predictor is duplicated into the predictor source code. This enables making decisions by using some application internal information that will be later computed by the application. It may also be possible to store the results of the computation of the predictor into memory, and to use them later in the application without recomputation. To implement this method, a trade-off must be made, between the amount of code duplicated and how early in the execution the current application scenario can be predicted. Usually, the earlier the better, but the prediction overhead must be kept under control.

Considering the runtime behavior, the predictors may be classified in:

- *Static*: All statistical information, except the scenario transition history, is collected at design time;
- *Adaptive*: Some statistical information is updated at runtime, based on the current execution (e.g. the scenario transition model).

4.4 Classification

The different classes of embedded systems (e.g. hard vs. soft real-time, single vs. multi-task applications) and the problem that must be solved, characterized for example by the resources of interest, lead to multiple possible criteria which can be used for scenario classification. In the following, we present the most important ones.

As the design methods for single and multi-task systems concentrate on different aspects, a classification of scenarios is based on their granularity:

- *Intra-task scenarios*, which appear within a sequential part of an application (i.e. a task);
- *Inter-task scenarios*, which represent operation modes of a multi-task application.

This classification can be seen as a hierarchy. Usually, there is a relation between intra-task and inter-task scenarios: the scenario in which a multi-task application is running is derived based on the scenarios in which each application task is currently running. Figure 6 depicts in a graphical way the possible relations between these two types of scenarios for an application with two tasks, each of them having two intra-task scenarios. An inter-task scenario could correspond to one or multiple combinations of the intra-task scenarios of each task.

Considering what the scenarios logically represent and how the switching between them is driven at runtime, two main categories of scenarios can be considered:

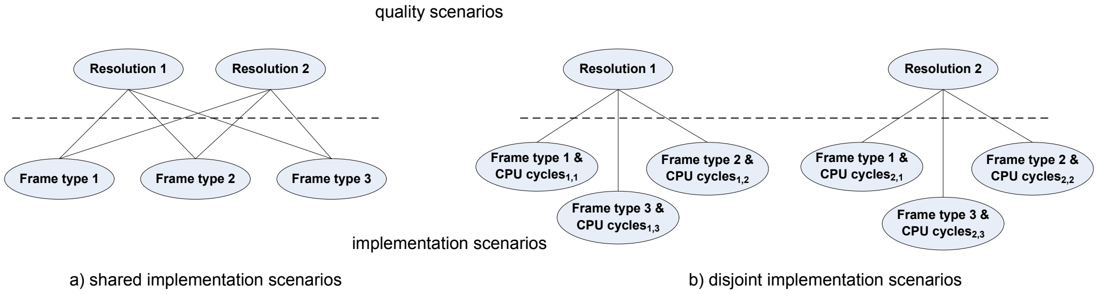
quality scenarios
Resolution 1
Resolution 2
Frame type 1
Frame type 2
Frame type 3
Frame type 1 & CPU cycles$_{1,1}$
Frame type 2 & CPU cycles$_{1,2}$
Frame type 3 & CPU cycles$_{1,3}$
Frame type 1 & CPU cycles$_{2,1}$
Frame type 2 & CPU cycles$_{2,2}$
Frame type 3 & CPU cycles$_{2,3}$
implementation scenarios
a) shared implementation scenarios
b) disjoint implementation scenarios

Figure 7: Possible relations between quality and implementation scenarios

- *Data flow driven* or *Implementation Scenarios*: Different actions in an application for the same quality parameters, selected at runtime based on the input data flow (e.g. type of streaming object to process). Each action has its own implementation within the application.

- *Event driven* or *Quality Scenarios*[5]: Different quality levels for the same functionality, which may be implemented as different algorithms or different quality parameters in the same algorithm. The scenario switching events are generated by the quality manager based on user requests or system status (e.g. battery level).

These scenarios can also be considered as a hierarchy. For different quality levels, an implementation scenario corresponding to the same application source code may require different amounts of resources.

An important aspect which differentiates the two types of runtime switches that may appear between scenarios is the different amounts of side-effects that are tolerable during them. Usually in case of implementation scenarios there must be no side-effects, and in case of quality scenarios different potential side-effects are acceptable.

Example: Let us consider a video decoder that has to display a new decoded video frame at every 50 milliseconds. From the implementation point of view there are different types of frames, which are decoded by different parts of the application code. If the implementation scenarios are characterized only by the source code used, then the implementation scenarios are shared between multiple quality scenarios as depicted by figure 7 (a). If the amount of computation cycles is considered as another resource of interest, then for the same frame type more cycles are needed for a higher image resolution than for a lower one. In this case, there will be no implementation scenarios shared between the quality scenarios (figure 7 (b)), although the source code is still shared. For the same video decoder, if it is implemented in a TV set, the changing of resolution (i.e., a change in quality scenario) can appear for example when a user wants to change from channel preview to channel watching, or when the image view mode is changed (e.g. from 4:3 to 16:9). In this case a side-effect of image flickering generated during system reconfiguration is acceptable. But when the user watches a channel and the application switches from an implementation scenario to another one, no side-effects visibly affecting the image can be accepted.

Even though in embedded systems design there is a large difference between the methods used for soft and hard real-time, we do not see any reason to characterize the scenarios based on this criterion. Obviously, for each of these two kinds of systems, not all the different methods presented above for each step of the methodology can be applied. For example, for hard real-time systems the scenario discovery may be done only using static analysis, and only detectors may be used to identify the current scenario at runtime, whereas for soft real-time systems predictors and statistical information from profilers may be used.

[5] or quality of service scenarios.

5 Literature Overview

In the following, we present a literature overview on both intra- and inter-task scenarios, concentrating on the implementation scenarios. Quality scenarios are out of the scope of this overview, but they can be found in papers related to quality of service (QoS), like [VKvBG95, GDvM$^+$03]. An exception is when there is no clear distinction in the presented paper between the implementation and quality scenarios. The section is divided in two parts, intra- and inter-task scenarios, as usually these are considered two separate categories in embedded system design.

5.1 Intra-task Scenarios

The most widely used language to specify single task applications for embedded systems is C. For these applications, the scenarios may be used to obtain a faster implementation (e.g. by a better memory mapping or by reducing the number of cache misses), an energy saving implementation, or a better estimation of the resources required by an application (e.g. for hard real-time applications, when the worst case execution time is required).

Our Work

Improved WCET Estimation [GSBC05b]: In this work the scenarios that incorporate correlations between different parts of the application are used to reduce the application estimated worst case execution cycles. The scenario identification was done using static analysis. As the scenarios are used only during design time analysis, no runtime predictor and code generation for scenarios were needed. An extensive report of this work is presented in [GSBC05a].

Scenario-Aware Voltage Scheduling [GBC05]: In this paper, we present how scenarios, which , as before, capture correlations between different parts of the application, can be applied on top of existing dynamic voltage scaling (DVS) techniques for hard real-time systems, making them more effective. The scenario identification is done by using a hybrid method and the application implementation is automatically generated. At runtime, a centralized/distributed detector finds the current scenario based on the values of different variables.

A Profiling Approach for Application Scenario Detection [GBC06]: This paper presents a method and a tool that can automatically detect the most important variables from an application and use them to define and dynamically predict scenarios, with respect to the necessary time budget, for soft real-time multimedia applications.

Related Work

Global Memory Optimization allowed by Code Duplication [PBV$^+$06, PCC05]: In this work, the authors, in a semi-automatic way, duplicate code in order to enable global loop transformations across data dependent conditions. They propose a hybrid technique for detecting application scenarios. They have a systematic way to combine these scenarios in sets and to build a Pareto curve [Par06, GBTO05] in 2D exploration space: memory usage vs. code size increase. The application implementation, including scenarios and the detector, is done by hand.

Energy Saving with Architectural and Frequency Adaptations for Multimedia Applications [HSA01]: In this work the authors identify for each manually detected scenario for a multimedia application the most energy efficient architecture configuration that can be used in order to meet the timing constraints. In their case the scenarios are defined for decoding an audio/video sample, and there are timing constraints for each scenario execution. The architectural adaptation is done only once, at the beginning of sample decoding. The considered architecture has a single processor with reconfigurable components (e.g. instructions issue window size, number and type of function units), and its supply voltage can be changed. The work was extended in [SHA02], allowing multiple hardware adaptations based on local decisions during a sample decoding. However, it is not clear how the scenarios are predicted at runtime.

Hardware Reconfiguration for Energy Reduction [VEB05b, VEB05a]: In this work the authors identify runtime phases of an application execution, and for each of them, reconfigure the hardware (in their case a simple processor) in order to consume less energy. Their phases are detected based on profiling, and are represented by a vector that captures how often each basic block from the program is executed. These phases are exploited at runtime by using a predictor. As the presented approach aims to be very general, it is not really suitable for multimedia applications. They do not have any way of incorporating knowledge about streaming objects in scenario discovery and runtime prediction.

Off-line Bitstream Analysis [BMP98, HCW05]: In this work the authors propose a platform dependent annotation of the bitstream during the encoding or before uploading it from a PC to a mobile system. As it is too time expensive to use a cycle accurate simulator to estimate the time budget necessary to decode each stream object, their approach uses a mathematical model to derive how many cycles are needed to decode each object. These works aim at a specific application, with a specific implementation, and require that each frame header contains a few parameters that characterize the computation complexity. None of them presents a way of detecting these parameters, all assuming that the designer will provide them, and will re-implement the application to take them into account.

Frame-Based Dynamic Voltage Scaling for an MPEG Decoder [PCDC02]: This paper describes a dynamic voltage and frequency scaling technique that reduces the energy consumption while maintaining a quality of service constraint for an MPEG decoder. Within the decoder the scenarios are manually selected. Furthermore, the final implementation of the decoder is also manually derived. The decoding of each frame is divided in two phases: a frame dependent one followed by a frame independent one. The frame independent phase will have the same computation requirements for all frames, so a *reactive* predictor is used in the application, after the frame dependent phase, to select the right voltage supply level in such a way that the timing constraints are met. In order to predict the frame dependent execution time, two types of probabilistic predictors are used: moving-average, where the next workload is predicted based on the average value of workloads during a predefined number of previous intervals, and weighted-average, where a weighted factor α is considered in calculating the future workload such that severe fluctuations of the workload are filtered out, resulting in a smaller average prediction error.

Cross-layer adaptive video coding to reduce energy [SAJ03]: In this paper the authors look for the most energy efficient configuration for encoding video on a mobile platform, exploring the balance between computation and compression efficiency. This means that a reduced compression efficiency, obtained by reducing the number of instructions used to perform compression, can actually reduce the total energy used to encode and transmit a video frame. This paper develops (i) an adaptive video encoder for general purpose processors that incorporate quality scenarios which trades computational complexity for compression efficiency to minimize total system energy, and (ii) a profiling based method for determining the best configuration for such an encoder when running on a voltage scaling aware processor. A probabilistic predictor is used at runtime to select the scenario used.

Summary and Open Issues

In the above mentioned papers, the main aim of using intra-task scenarios is to reduce the computation requirements and the energy consumption. Usually these approaches lack a tool-flow infrastructure, most of the work being done manually by the designer. However, most of them could be automatized into compilers specific for streaming applications.

Besides computation cycles and processor energy, other types of resources should be also considered when scenarios are defined. Current developments in multimedia embedded systems show that chips are becoming memory dominated (estimated 90% in 2010) [OS00] for two reasons. Firstly, logic scales faster with chip technology than memory. Secondly, current multimedia applications require increasingly more memory. This prediction shows that memory usage will become a very costly factor for the system, from size, energy and cost point of view. Thus, there should be more research focussed on optimizing memory usage based on scenarios.

As portable multimedia embedded systems have became pervasive in the past decade, the video and audio standards have to start taking into account their requirements. The most important one is energy efficiency. It can be met by incorporating in multimedia streams information that characterizes the amount of required resources to decode the next streaming object. Moreover, the standards definitions should not concentrate only on the compressed data size reduction, but also on the amount of computation necessary to decode the resulted encoded objects.

5.2 Inter-task scenarios

There is some agreement on a common way to express a single task application. However, for multi-task applications there is not a well recognized representation model. The used ones appear like task graphs, and especially for signal processing and multimedia applications like data flow graphs. In this section we discuss the models that could incorporate or already incorporate implementation scenarios, and how these scenarios have been already used in different steps of the multi-task embedded systems design trajectory. The use in fact turns out to be limited.

5.2.1 Application Models

In the past years, multiple data flow models suitable for multimedia applications were researched. However, an open point is which ones already incorporate, or can easily incorporate, an application described by both scenarios and the transitions between them. Possible candidates for the moment are:

- *A set of Synchronous Data Flow Graphs*: The Synchronous Data Flow (SDF) Model [LM87] is the most analyzable data flow model. It allows checking for many correctness and performance properties, but it lacks the support for expressing any form of dynamism, like scenarios. The easiest way of representing an application together with its scenarios, is to represent each scenario as a separate SDF, and to find a way to model the transitions between these scenarios outside the model.

- *Hierarchical Finite State Machines (FSM) and the Synchronous Data Flow Model [Lee00, LYC02]* extend the SDF model, adding a way to represent the application dynamism via an FSM. Although the authors do not explicitly mention scenarios, this model of computation can be use to extend the previous idea of representing each scenario like an SDF. These scenarios are linked together by a scenario switching detector/predictor, represented as an FSM. The drawbacks of this model of computation are that it does not incorporate the correlations between the scenarios (e.g. a task from one scenario which is identical to a task from another scenario), and that it looses much of the analyzability properties of an SDF model.

- *Boolean [Buc93], Integer Controlled [Buc94b], Parameterized [BB01] or Dynamic [Buc94a] Data Flow Models* are extensions of the SDF model, which allow different kinds of dynamism. This dynamism reduces their analyzability properties. Even if an application that contains scenarios may be represented using these models, the scenarios and the runtime predictor will be hidden in the model, meaning that a way of discovering them must be developed.

- *Kahn Process Networks (KPN) [Kah74]* are a developer friendly model generalizing SDF that can capture the possible data-dependent dynamism of a multi-task system, including also some use-case, quality and implementation scenarios. This model can not be used to capture non-deterministic or reactive/interactive behaviors (e.g. events). Moreover, the evaluation of its correctness and performance is in general undecidable. Furthermore, like the other SDF extensions, if an application that incorporates multiple scenarios is represented by it, these scenarios are hidden in the model.

- *Scenario-Aware Data Flow Models (SADF) [TGB+06]* are a design-time analyzable stochastic generalization of SDF, which can capture several dynamic aspects of modern streaming applications by incorporating application scenarios. The scenario and the runtime predictor are explicitly described in the model, no extra need for discovery being necessary. Moreover, analysis of correctness as well as long-run average and worst-case performance are decidable. SADF combines both analyzability and explicit representation of scenarios. The only current drawback is that not all possible forms of dynamism (e.g. interactions with external events) can be represented with it.
- *Reactive Process Networks(RPN) [GB04]* are a very general model of computation. It combines the basic principles of Kahn process networks and state machines in a compositional way. It can handle all forms of dynamism, allowing to make scenarios and predictors explicit. However, in its most generic form, it is not analyzable.

If a multi-task application is developed from scratch, and the system designer wants to take advantage of application scenarios during the system implementation, we strongly recommend to represent the application using a model that makes these scenarios, the predictor used to switch between them at runtime, and the relations between the structure of scenarios (e.g. multiple scenarios may share the same tasks) explicit. Also, the required analyzable properties have to be taken into account when selecting the model.

5.2.2 ES Design Trajectory

The intra-task application scenarios have not been extensively used in the design trajectory for multi-task embedded systems. Usually, this trajectory maps a multi-task application to a single- or multi-processor system-on-chip. Typically, it consists of many steps, like task binding, task scheduling, communication mapping. In this section we present different results obtained by exploiting application scenarios during these steps.

Estimation of Execution Times of On-chip Multiprocessor Stream-Oriented Applications [PBP+05]: This paper aims at estimating the execution time of stream-oriented applications mapped on a multi-processor on-chip. For this kind of systems the pipelined decoding of sequential streaming objects has a high impact on achieving the required throughput. The application is modeled as a homogenous synchronous data flow graph, which is similar to an SDF with a few restrictions, but has a larger number of analyzable properties. In the application, the loop of interest (i.e. the loop which is executed over and over again) is taken, and within it the scenarios are manually defined based on the different execution workloads of tasks. The authors propose an accurate execution time estimation method that supports parallel and pipelined decoding of streaming objects, taking into account scenario switching.

Cost-Efficient Mapping of Dynamic Concurrent Tasks [YMW+03, Yan04, Chu03]: The authors used scenarios to capture the data-dependent dynamic behavior inside a thread, to better schedule a multi-thread application on a heterogenous multi-processor architecture. As the authors considered also the possibility of changing the voltage level for each individual processor, their work can be considered as combining inter-task voltage scheduling with scenarios. In this work, both the scenario identification and the application rewriting to use scenarios at runtime were done by hand. The scenarios were identified based on the designer's knowledge about different possible runtime behaviors. The model used is a set of task graphs, having a task graph for each scenario. Moreover, they did not model the transitions from one scenario to another.

Inter-Task Dynamic Voltage Scaling [LYC02]: In this work, the application is written using a combination of a hierarchical finite state machine (FSM) with a synchronous data flow model (SDF). The FSM represents the scenarios' runtime detector. The scenarios are identified by the designer and they are already described in the model. They showed that by writing the application in this model, the scenario knowledge can be used to save energy when mapping the application on one processor.

Mapping Multiple Scenarios onto Networks on Chips (NoC) [MCR+06a]: This paper motivates

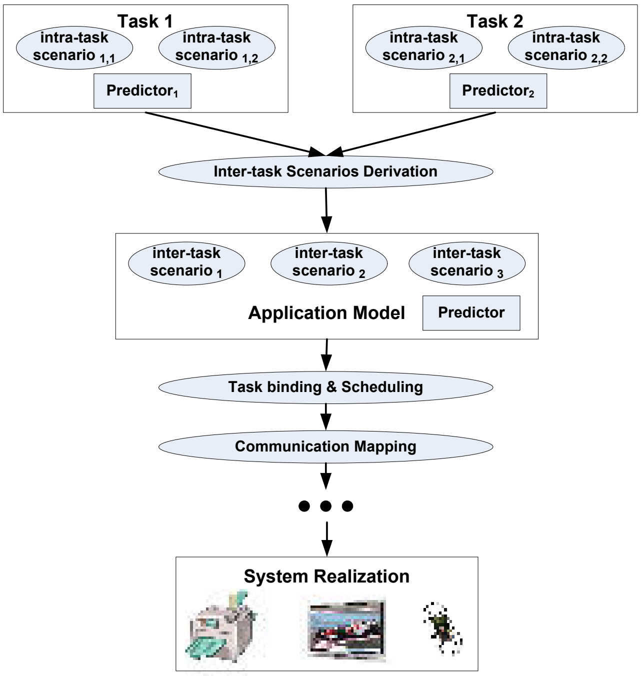

Figure 8: Required design flow for multi-task embedded systems

the need of considering multiple scenarios during the design process. The scenarios are characterized by different communication requirements (such as different bandwidth, latency constraints) and traffic patterns. The paper presents a method to map an application to a NoC architecture, satisfying the design constraints of each individual scenario. This approach concentrates on the communication aspect of the application mapping. It allows dynamic network reconfiguration across different scenarios. As the over-estimation of the worst case communication is very large, this method performs poorly on systems where the traffic characteristics of scenarios are very different or when the number of scenarios is large. In [MCR$^+$06b] the method was extended to work for these cases.

Characterizing Workload Correlations in Multi Processor Hard Real-Time Systems [WT05]: This paper extends a existent method for performance analysis of hard-real time systems based on Real-Time Calculus, taking into account correlations that appear between different components of the system. These correlations form the application scenarios. The authors present only how these scenarios could be modeled in their high level modeling/analytical approach, but no way to discover scenarios and no prediction mechanism was considered.

5.2.3 Summary and Open Issues

The usage of inter-task scenarios within the multi-task embedded system design trajectory has not been extensively explored yet. A design flow, like the one presented in figure 8, will help in producing cheaper systems, as it was proved by [MCR$^+$06b] for the case of mapping the inter-task communication on a network on-chip, which is one of the design trajectory steps. The flow should start from the intra-task scenarios extracted for each application task, and based on them derive the inter-task application scenarios. Then, these scenarios are used in decision making along the design trajectory, like in task binding and scheduling.

6 Conclusions

In this report we distinguished between different scenarios (use-case and application scenarios) that may appear in an embedded system design process. We focused on the application scenarios which, with the exception of quality scenarios, were not explored and used much previously, especially in multi-task systems. We gave a definition, a classification and a basic methodology of using them in the design process. The results of using application scenarios, presented in the literature overview, show that the application scenarios can improve the quality of the final system (e.g. lower energy consumption, cost or size). Considering this, we strongly recommend the exploitation of application scenarios in all the steps of design trajectory.

References

[BB01] B. Bhattacharya and S.S. Bhattacharyya. Parameterized dataflow modeling for DSP systems. *IEEE Transactions on Signal Processing*, 49(10):2408–2421, October 2001.

[BMP98] Andy C. Bavier, A. Brady Montz, and Larry L. Peterson. Predicting MPEG execution times. *ACM SIGMETRICS Performance Evaluation Review archive*, 26(1):131–140, June 1998.

[Buc93] Joseph Tobin Buck. *Scheduling Dynamic Dataflow Graphs with Bounded Memory Using the Token Flow Model.* PhD thesis, University of California, 1993.

[Buc94a] Joseph T. Buck. A dynamic dataflow model suitable for efficient mixed hardware and software implementations of DSP applications. In *Proc. of the 3rd Int. Workshop on Hardware/Software Co-design*, pages 165–172, Grenoble, France, 1994.

[Buc94b] Joseph Tobin Buck. Static scheduling and code generation from dynamic dataflow graphs with integer valued control streams. In *Proc. of the 28th Asilomar Conf. on Signals, Systems, and Computers*, pages 508–513. IEEE, 1994.

[Car95] John M. Carroll, editor. *Scenario-based design: envisioning work and technology in system development.* John Wiley & Sons Inc, NY, USA, 1995.

[Chu03] Wong Chun. *Design-Time Sub-Task Scheduling for Embedded Multimedia and Telecom Systems.* PhD thesis, Catholic University of Leuven, Belgium, September 2003.

[Dou04] Bruce Powel Douglass. *Real Time UML: Advances in the UML for Real-Time Systems.* Addison Wesley Publishing Company, 2004.

[Fow03] Martin Fowler. Use cases. In *UML Distilled: A Brief Guide to the Standard Object Modeling Language, Third Edition*, chapter 9, pages 99–106. Addison Wesley Publishing Company, 2003.

[GB04] M.C.W. Geilen and T. Basten. Reactive process networks. In *Proc. of the 4th ACM Int. Conf. on Embedded Software*, pages 137–146, Pisa, Italy, 2004. ACM Press.

[GBC05] Stefan Valentin Gheorghita, Twan Basten, and Henk Corporaal. Intra-task scenario-aware voltage scheduling. In *Proc. of the Int. Conf. Compilers, Architecture and Synthesis for Embedded Systesms (CASES)*, pages 177–184, San Francisco, CA, USA, September 2005. ACM Press.

[GBC06] Stefan Valentin Gheorghita, Twan Basten, and Henk Corporaal. Profiling driven scenario detection and prediction for multimedia applications. In *(accepted) at the Int. Conf. on Embedded Computer Systems: Architectures, MOdeling, and Simulation (IC-SAMOS)*, Samos, Greece, 2006. IEEE Computer Society Press.

[GBTO05] M.C.W. Geilen, T. Basten, B.D. Theelen, and R.H.J.M. Otten. An algebra of pareto points. In *Proc. of the 5th Int. Conf. in Application of Concurrency to System Design ACSD*, pages 88–97, St Malo, France, 2005. IEEE Computer Society Press.

[GDvM+03] Kees Goossens, John Dielissen, Jef van Meerbergen, Peter Poplavko, Andrei Radulescu, Edwin Rijpkema, Erwin Waterlander, and Paul Wielage. Guaranteeing the quality of services in networks on chip. In *Networks on chip*, chapter 4, pages 61–82. Kluwer Academic Publishers, Hingham, MA, USA, 2003.

[GSBC05a] Stefan Valentin Gheorghita, Sander Stuijk, Twan Basten, and Henk Corporaal. Sharper WCET upper bounds using automatically detected scenarios. Technical Report ESR-2005-04, Technical University of Eindhoven, Electrical Engineering Department, Electronic Systems Group, Eindhoven, Netherlands, March 2005.

[GSBC05b] Stefan Valentin Gheorghita, Sander Stuijk, Twant Basten, and Henk Corporaal. Automatic scenario detection for improved wcet estimation. In *Proc. of the 42nd Design Automation Conf. DAC*, pages 101–104, Anaheim, CA, USA, June 2005. ACM Press.

[HCW05] Yicheng Huang, Samarjit Chakraborty, and Ye Wang. Using offline bitstream analysis for power-aware video decoding in portable devices. In *Proc. of the 13th ACM Int. Conf. on Multimedia*, pages 299–302, Singapore, November 2005.

[HSA01] Christopher J. Hughes, Jayanh Srinivasan, and Sarita V. Adve. Saving energy with architectural and frequency adaptations for multimedia applications. In *Proc. of the 34th Annual ACM/IEEE Int. Symposium on Microarchitecture (MICRO)*, pages 250–261. IEEE Computer Society Press, 2001.

[IEE00] IEEE standard 1471: Recommended practice for architectural description of software-intensive systems, 2000.

[Ion05] Mugurel Theodor Ionita. *Scenario-based system architecting: a systematic approach to developing future-proof system architectures.* PhD thesis, Technische Universiteit Eindhoven, The Netherlands, May 2005.

[Jac95] Ivar Jacobson. The use-case construct in object-oriented software engineering. In *Scenario-Based Design: Envisioning Work and Technology in System Development*, chapter 12, pages 309–336. John Wiley & Sons, NY, USA, 1995.

[Kah74] G. Kahn. The semantics of a simple language for parallel programming. In *Proc of Int. Forum on Information Processing*, pages 471–475, North-Holland, 1974.

[Lee00] B. Lee. *Specification and Design of Reactive Systems.* PhD thesis, Memorandum UCB/ERL M00/29, EECS Department, University of California, Berkeley, August 2000.

[LM87] E.A. Lee and D.G. Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235–1245, September 1987.

[LYC02] Sunghyun Lee, Sungjoo Yoo, and Kiyoung Choi. An intra-task dynamic voltage scaling method for SoC design with hierarchical FSM and synchronous dataflow model. In *Proc. of the Int. Symposium on Low Power Electronics and Design*, pages 84–87, Monterey, CA, USA, August 2002.

[MCR+06a] Srinivasan Murali, Martijn Coenen, Andrei Radulescu, Kees Goossens, and Giovanni De Micheli. Mapping and configuration methods for multi-use-case networks on chips. In *Proc. of the Asia South Pacific Design Automation Conf. (ASPDAC)*, pages 146–151, Yokohama, Japan, 2006. ACM Press.

[MCR+06b] Srinivasan Murali, Martijn Coenen, Andrei Radulescu, Kees Goossens, and Giovanni De Micheli. A methodology for mapping multiple use-cases onto networks on chips. In *Proc. of the Design, Automation and Test in Europe (DATE)*, Munich, Germany, 2006.

[OS00] Ralph H.J.M. Otten and Paul Stravers. Challenges in physical chip design. In *Proc. of the IEEE/ACM Int. Conf. on Computer-aided Design (ICCAD)*, pages 84–92. ACM press, 2000.

[Par06] Vilfredo Pareto. *Manuale di Economia Politica*. Piccola Biblioteca Scientifica, Milan, 1906. Translated into English by A.S. Schwier (1971), Manual of Political Economy, MacMillan, London.

[Pau05] JoAnn M. Paul. Scenario-oriented design for single chip heterogeneous multiprocessor. In *Proc. of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 10*, page 227b, 2005.

[PBP+05] P. Poplavko, T. Basten, M. Pastrnak, J. van Meerbergen, M. Bekooij, and P. de With. Estimation of execution times of on-chip multiprocessors stream-oriented applications. In *Proc. of the 3rd ACM & IEEE Int. Conf. in Formal Methods and Models for Codesign (MEMOCODE 2005)*, pages 251–252, Verona, Italy, July 2005.

[PBV+06] Martin Palkovic, Erik Brockmeyer, Peter Vanbroekhoven, Henk Corporaal, and Francky Catthoor. Systematic preprocessing of data dependent constructs for embedded systems. *Journal of Low Power Electronics*, 2(1):9–17, 2006.

[PCC05] Martin Palkovic, Henk Corporaal, and Francky Catthoor. Global memory optimisation for embedded systems allowed by code duplication. In *Proc. of the 9th Int. Workshop on Software and Compilers for Embedded Systems*, Dallas, Texas, September 2005.

[PCDC02] Massoud Pedram, Wei-Chung Cheng, Karthik Dantu, and Kihwan Choi. Frame-based dynamic voltage and frequency scaling for a MPEG decoder. In *Proceedings of IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD '02)*, pages 732–737, San Jose, CA, USA, November 2002.

[RC02] Mary Beth Rosson and John M. Carroll. Scenario-based design. In *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications*, chapter 53, pages 1032–1050. Lawrence Erlbaum Associates, Mahwah, NJ, 2002.

[SAJ03] Daniel Grobe Sachs, Sarita V. Adve, and Douglas L. Jones. Cross-layer adaptive video coding to reduce energy on general-purpose processors. In *Proc. of IEEE Intl. Conf. on Image Processing*, pages 109–112. IEEE Computer Society Press, 2003.

[SHA02] Ruchira Sasanka, Christopher J. Hughes, and Sarita V. Adve. Joint local and global hardware adaptations for energy. *ACM SIGARCH Computer Architecture News*, 30(5):144–155, 2002.

[TGB+06] B.D. Theelen, M.C.W. Geilen, T. Basten, J.P.M. Voeten, S.V. Gheorghita, and S. Stuijk. A scenario-aware data flow model for combined long-run average and worst-case performance analysis. 2006.

[VEB05a] Frederik Vandeputte, Lieven Eeckhout, and Koen De Bosschere. A detailed study on phase predictors. In *Proc. of the 11th Inte. Euro-Par Conf.*, pages 571–581, Lisbon, Portugal, August 2005.

[VEB05b] Frederik Vandeputte, Lieven Eeckhout, and Koen De Bosschere. Offline phase analysis and optimization for multi-configuration processors. In *Proc. of the 5th Int. Workshop in*

Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), pages 202–211, Samos, Greece, July 2005.

[VKvBG95] Andreas Vogel, Brigitte Kerherve, Gregor von Bochmann, and Jan Gecsei. Distributed multimedia and QoS: a survey. *IEEE Multimedia*, 2(2):10–19, April 1995.

[WT05] E. Wandeler and L. Thiele. Characterizing workload correlations in multi processor hard real-time systems. In *Proc. of the 11th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 46–55, San Francisco, USA, 2005.

[Yan04] Peng Yang. *Pareto-Optimization based Run-Time Task Scheduling for Embedded Systems*. PhD thesis, Catholic University of Leuven, Belgium, September 2004.

[YMW+03] Peng Yang, Paul Marchal, Chun Wong, Stefaan Himpe, Francky Catthoor, Patrick David, Johan Vounckx, and Rudy Lauwereins. *Multi-Processor Systems on Chip*, chapter Cost-efficient mapping of dynamic concurrent tasks in embedded real-time multimedia systems. Morgan Kaufmann, 2003.