The Disappearing Computer

"The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it." Mark Weiser, in [1]

Embedded systems

The personal computer as we know it is rapidly disappearing. Computers are more and more often embedded in everyday appliances, professional and industrial equipment, and in our surroundings. Everything in our society – communication, the healthcare system, the financial system, transportation, the electrical infrastructure – is driven by, in fact, depends on, embedded computers. Embedded computing is what makes our society smart. Reliability, performance, energy efficiency and cost of embedded computer systems are crucial. Model-driven development of embedded systems is needed to achieve the required product quality in a cost-effective way. To date, we lack the appropriate modeling abstractions to predict system properties of embedded systems early during development.

The automatic car park just smashed your car. Sorry. [Volkskrant, 25 January 2005, vk.nl]

Challenges

One of the research themes in the Electronic Systems group is model-driven design for embedded systems. Challenges we are trying to address: Can we accurately predict performance aspects such as latency and throughput without resorting to a prototype? Can we predict power consumption? And reliability? How do we automatically synthesize system implementations that satisfy performance and reliability requirements within given power budgets? How do we effectively explore the huge space of design alternatives? Can we characterize the trade-offs between performance, reliability, energy-efficiency and costs? Can we develop systems that work reliably under different , changing circumstances?

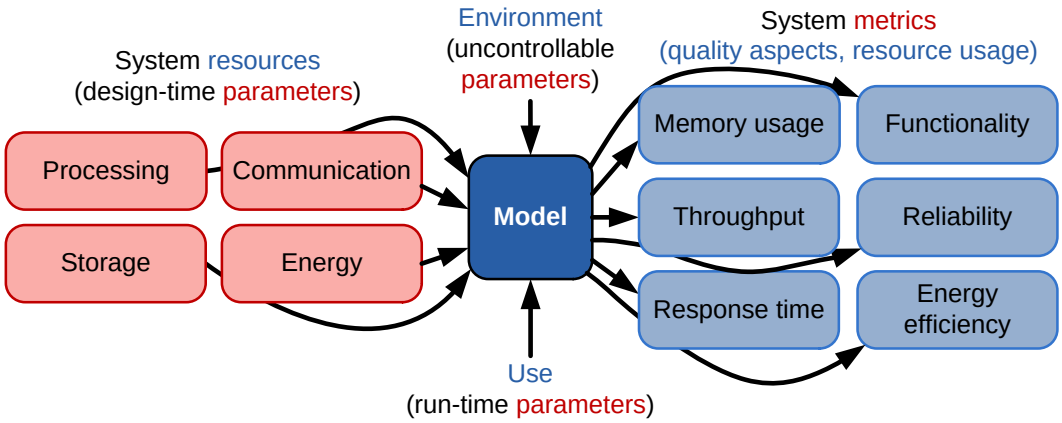
Trends

Embedded systems are rapidly evolving from closed, sequential, monolithic systems performing a single function to open, parallel, networked systems performing a multitude of functions and applications in

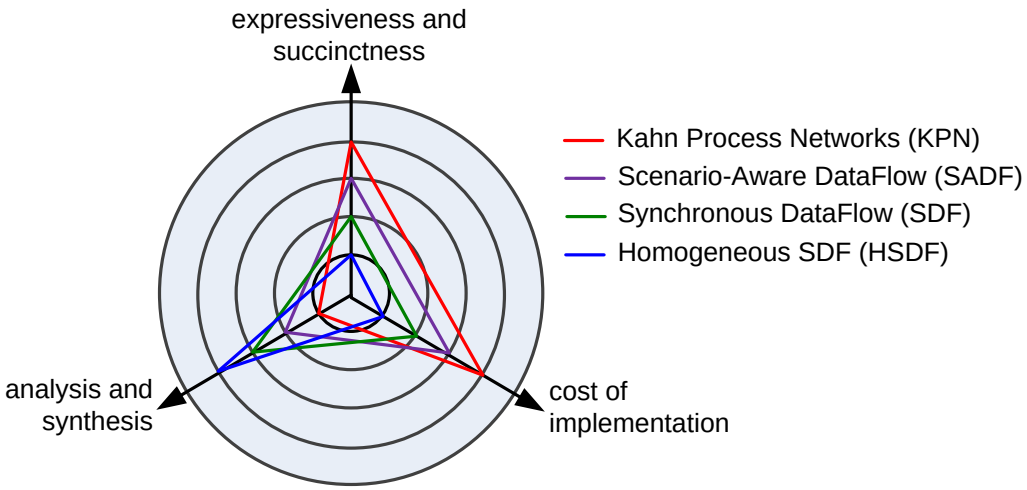
different combinations. The evolution of mobile phones into smart phones is a prominent example of this trend. Predicting performance or power dissipation of such systems, guaranteeing that applications always meet their deadlines, and guaranteeing the proper functioning under all circumstances is challenging. Other trends that complicate matters are the increasing variability in the properties of produced hardware, and the fact that system performance depends more and more often on the data being processed. These trends imply that the nature of embedded computers is changing, and that the complexity of both the embedded systems themselves and their development trajectory is growing rapidly.

Computational models

A computational model relates system parameters, like amount and type of processing and storage resources, to system metrics, like throughput or energy efficiency. Models are used both for analysis – deriving properties of a system – and for synthesis – constructing system implementations satisfying certain properties. Computational modeling plays a crucial role in addressing the mentioned challenges and in coping with the growing complexity of embedded systems and their development. We need appropriate modeling abstractions with accompanying analysis and synthesis techniques. We further need to embed these techniques in a model-driven development process that is efficient and that leads to high-quality results. It is very difficult though to combine good expressive modeling power with powerful analysis and synthesis techniques while keeping the cost of model-driven implementation low.



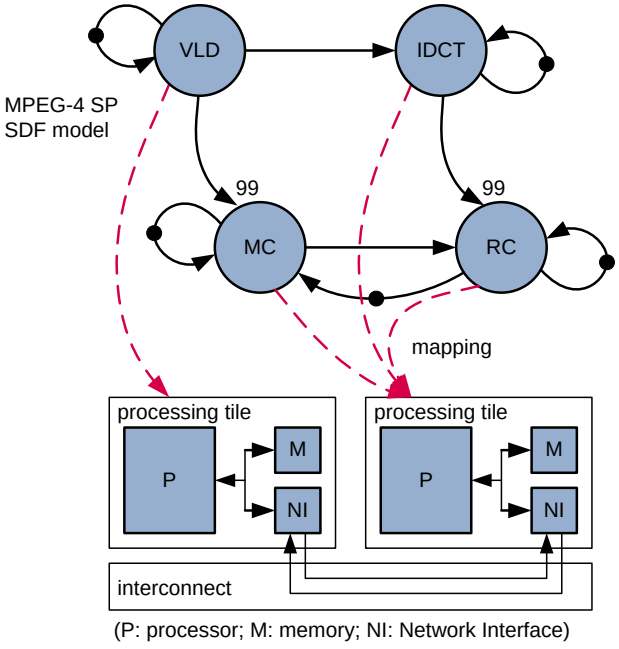
A computational model relates parameters to metrics of interest.



Conflicting requirements on computational models.

An MPEG-4 example

Consider an MPEG-4 SP video decoder application running on a multiprocessor system-on-chip. The figure below shows a synchronous dataflow (SDF) model. It captures the required computations in four so-called actors – VLD, IDCT, MC and RC. The computations themselves are abstracted away, and only the worst-case actor execution times on a specific type of processor are taken into account. The edges between the actors, called channels, capture data and control dependencies. Communication occurs through tokens, abstract units of data or control. Each actor consumes upon execution a fixed number of tokens from each of its incoming channels and produces a fixed number of tokens on each of its outgoing channels. Channels are annotated with these numbers, called rates, where a rate of 1 is omitted. A VLD actor execution, for example, produces one token on its channel to MC, whereas MC needs 99 tokens from this channel in order to execute. Channels may contain initial tokens, the black dots in the figure; this allows capturing delayed dependencies.



analysis result	
throughput guarantee	306 (frames per second)

execution time (µs)	
VLD	33
IDCT	14
MC	325
RC	292

synthesis result	
throughput guarantee	15 (frames per second)
processor capacity	88 %
memory capacity	254 KB
interconnect bandwidth	33 %

A synchronous dataflow model of an MPEG-4 SP decoder running on a multiprocessor.

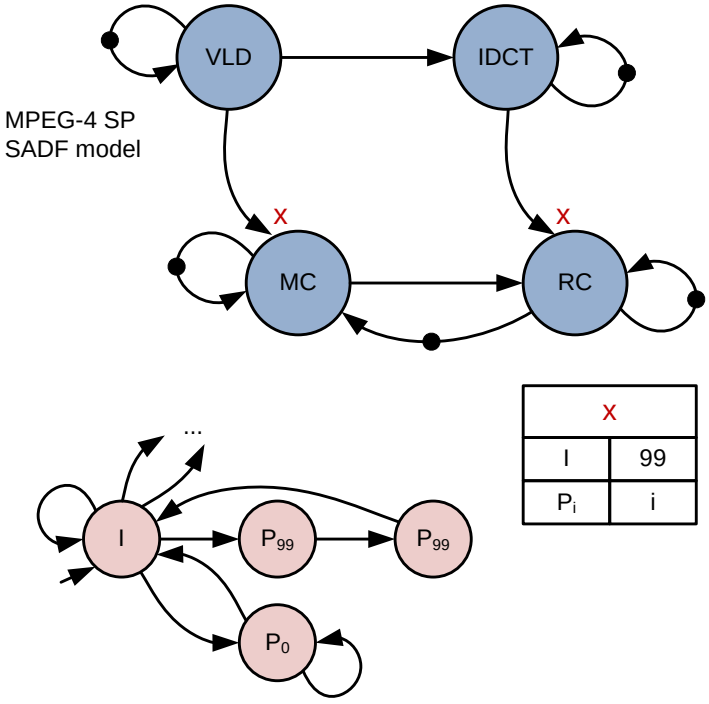
An SDF model can be analyzed for various properties, among others the maximal achievable throughput. An SDF model executes in iterations. One iteration returns the model to its original state. In the example, an iteration consists of 99 VLD and IDCT executions combined with one execution of MC and RC. One iteration corresponds to the decoding of a single video frame. When not yet taking the illustrated mapping onto the 2-processor platform into account, it is possible to guarantee a throughput of 306 frames per second.

Any mapping on a resource-constrained platform will limit performance, because of resource sharing, communication delays, etc. The earlier analysis result for the MPEG-4 decoder shows that performance is good enough to investigate a mapping on a platform with only limited resources. Using an SDF model and a platform description giving the number and speed of processors, the size and access latency of memories, and the speed and capacity of the interconnect, a mapping can be synthesized for any given throughput constraint. The figure shows the resulting mapping for the MPEG-4 decoder for a

2-processor platform and a throughput guarantee of 15 frames per second. The trajectory also gives the required processor, memory and interconnect capacities.

Good models

The SDF model and accompanying synthesis trajectory illustrate the most important benefit of model-driven development: an end result with a guaranteed quality, a throughput guarantee in the MPEG-4 example. Other potential benefits are energy- and cost efficiency and faster development, resulting in a shorter time-to-market. An important question remains however. How good are the results? It is not clear whether or not better results could have been achieved through a manual implementation trajectory, or through the use of different models.



X	
I	99
Pi	i

execution time (□s)		
VLD	P_0	0
	other	33
IDCT	P_0	0
	other	14
MC	I,P_0	0
	P_{30}	75
	P_{40}	121
	P_{50}	158
	P_{60}	196
	P_{70}	221
	P_{80}	258
	P_{99}	325
RC	I	292
	P_0	0
	P_{30},P_{40},P_{50}	208
	P_{60}	250
	P_{70},P_{80},P_{99}	267

An SADF model of the MPEG-4 SP decoder, capturing data-dependent dynamics.

An important characteristic of modern multimedia applications is that the workload depends on the data being processed. In video decoding, for example, the amount of movement of objects between subsequent frames has a large impact on the processing workload and the amount of data flowing through the processing pipeline. An SDF model abstracts from the workload dynamics by assuming worst-case actor execution times and worst-case data rates. This may lead to overly conservative results, leading to unnecessary resource reservations, in turn implying extra costs and energy dissipation.

Workload dynamics can be captured in a Scenario-Aware DataFlow (SADF) model. Such a model essentially defines a number of scenarios. For the MPEG-4 example, nine scenarios for decoding a frame can be defined (I, P_0, P_{30}, …, P_{80}, P_{99}), based on the coding being used (I or P) and the amount of motion between frames. A state machine defines the possible orders in which scenarios may occur; each scenario is captured by an SDF model. In the example, the actor execution times and the consumption rates of actors MC and RC may differ between scenarios; the structure of the SDF model remains constant. An important observation is that the worst-case actor execution times for different actors

occur in different scenarios. This provides room for improvement when compared to the SDF model of the decoder. With the synthesis trajectory described in [2], a mapping can be synthesized for the same 2-processor platform that we have seen before. The results are given in the table below. For the same throughput guarantee, processor capacity reservations can be reduced by 70% when compared to the result of SDF synthesis. A similar exercise comparing SDF and SADF synthesis for an MP3 decoder shows substantial savings for memory and interconnect usage. These savings are obtained purely by using more appropriate models. The functionality and performance of the end result is not affected.

MPEG-4 SP				MP3
metric	SDF	SADF	improvement	improvement
througput guarantee	15 (frames per second)	15 (frames per second)	NA	NA
processor capacity	88 %	26 %	70 %	-
memory capacity	254 KB	254 KB	-	21 %
interconnect bandwidth	33 %	33 %	-	23 %

Comparison between SDF synthesis and SADF synthesis.

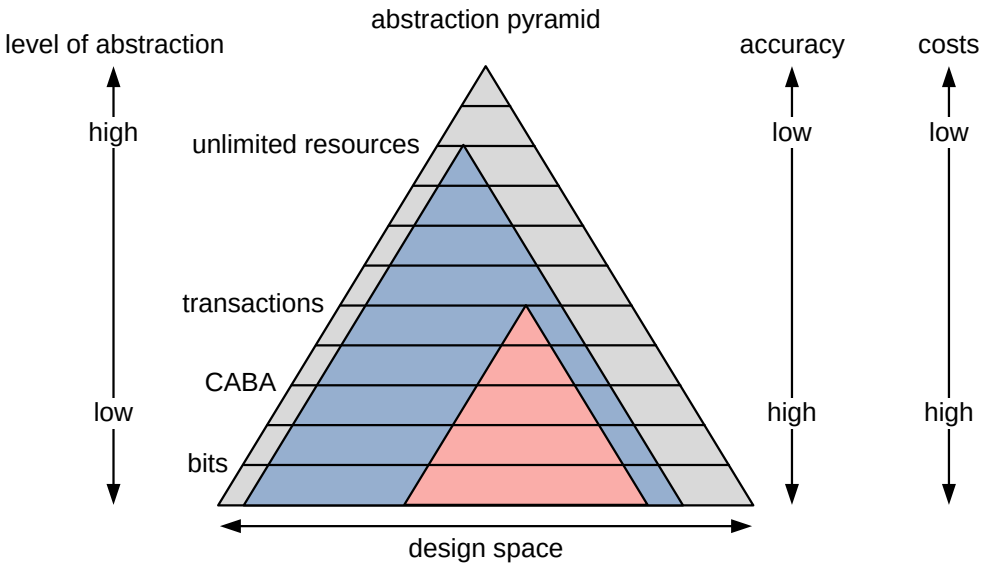
When comparing SDF and SADF, we see that SADF is more expressive than SDF. A broader range of analysis and synthesis techniques is available for SDF though and the cost of model-driven implementation is lower. Aspects like concurrency, data-dependent behavior and pipelined execution of iterations make analysis and synthesis of SADF models challenging. Nevertheless, SADF gives, overall, the best result for the MPEG-4 and MP3 examples. This illustrates the importance of choosing the right modeling techniques. The comparison between SDF and SADF further illustrates the conflicts between model expressiveness and compactness, on the one hand, and the availability of analysis and synthesis techniques and the cost of implementation, on the other hand.

SADF has been developed in the Electronic Systems group. Its development is illustrative for the type of research in computational modeling that we are doing. We aim to find the appropriate modeling abstractions and accompanying analysis and synthesis techniques to address the challenges in embedded systems development. All results are laid down in tooling and made available to the community at large. Industrial adoption is our ultimate goal.

Complexity

Another important challenge is the ever growing complexity of embedded systems and their development. The number of design alternatives for an embedded system is huge; each design alternative comes with a different trade-off in terms of functionality, costs, performance, energy efficiency, reliability, etc. How to find the right design alternative? The answer to this question boils down to making the right decisions at the right abstraction level. At higher abstraction levels, a larger part of the design space can be explored at a reasonable cost; model predictions of system properties will have limited accuracy though, so one has to be careful to make only those decisions for which the predictions are sufficiently precise. To date, the model abstractions for the lower abstraction levels of embedded system design are widely accepted; there is no consensus though about the right higher-level

abstractions for open, interacting, multiprocessor systems. Also the combination of the appropriate modeling, analysis and synthesis techniques into model-driven development trajectories for different types of systems is still open. Standardized model-driven development trajectories accepted by industry are crucial for the development of reliable, high-quality, cost-effective embedded systems. If we do not get a grip on complexity, the envisioned smart society will not materialize.



The abstraction pyramid.

Predictable systems

To keep complexity manageable, development processes need to be improved through structured component-based design. Furthermore, predictability of system properties at all levels of abstraction is a prerequisite. It is clear that further developments in computational modeling will lead to improved predictability. However, also the embedded systems themselves may be designed for better predictability. Nowadays, cost, performance and energy consumption are the drivers for design. Predictability should be added. Models and systems should be developed hand-in-hand. The most prominent example of such a co-development is the digital abstraction. Sequential digital circuits can be modeled, analyzed and synthesized with state-machine models. The clock that synchronizes state transitions ensures that models and realizations fit. Synchronously clocked circuits are still dominant today, despite the fact that synchronous clocks have important disadvantages, such as high power dissipation and high wiring cost. Synchronous clocks have been invented to manage complexity and to achieve predictability. Predictability should again become a key driver in embedded system design.

Understanding

Computational modeling will play a crucial role in the further development of a smart society driven by embedded computers. Computers will disappear. They will become part of our natural environment, as envisioned two decades ago by Mark Weiser. The most important benefit of modeling is that it leads to understanding.

“An understanding of the natural world and what's in it is a source of not only a great curiosity but great fulfillment.” David Attenborough

Further reading

Readers interested in the original ubiquitous computing vision laid out by Mark Weiser may have a look at [1]. Ref. [2] describes a multiprocessor synthesis trajectory for Scenario-Aware Dataflow, whereas [3] surveys analysis and synthesis techniques for SADF. The last two papers give many pointers for further reading about dataflow modeling, analysis and synthesis.

1. M. Weiser. The Computer for the 21st Century. Scientific American, 165(3):94–104, 1991, Reprinted in IEEE Pervasive Computing, Jan-Feb 2002, p. 19-25.
2. S. Stuijk, M. Geilen, T. Basten. *A Predictable Multiprocessor Design Flow for Streaming Applications with Dynamic Behaviour.* In *Digital System Design, 13th EUROMICRO Conference, DSD 2010*, Proceedings, pages 548-555. IEEE CS Press, 2010.
3. S. Stuijk, M. Geilen, B. Theelen, T. Basten. *Scenario-Aware Dataflow: Modeling, Analysis and Implementation of Dynamic Applications*. In *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, IC-SAMOS 11*, Proceedings, pages 404-411. IEEE CS Press, 2011.

Twan Basten, professor of computational models, Electronic Systems group

This article is a short version of my inaugural lecture, given on April 15, 2011