Model-Driven Design-Space Exploration for Software-Intensive Embedded Systems*
(extended abstract)

Twan Basten[1,2], Martijn Hendriks[1], Lou Somers[2,3], and Nikola Trčka[4]

[1]Embedded Systems Institute, Eindhoven, the Netherlands
[2]Eindhoven University of Technology, Eindhoven, the Netherlands
[3]Océ Technologies B.V., Venlo, the Netherlands
[4]United Technologies Research Center, East Hartford, CT, USA
http://dse.esi.nl

Abstract. Software plays an increasingly important role in modern embedded systems, leading to a rapid increase in design complexity. Model-driven exploration of design alternatives leads to shorter, more predictable development times and better controlled product quality.

1 Motivation

Industries in the high-tech embedded domain (including for example professional printing, lithographic systems, medical imaging, automotive, etc.) are facing the challenge of rapidly increasing complexity of next generations of their systems. Ever more functionality is being added, user expectations regarding quality and reliability increase, an ever tighter integration between the physical processes being controlled and the embedded hardware and software is needed, and technological developments push towards networked, multi-processor and multi-core platforms. The added complexity materializes in the software and hardware embedded at the core of the systems. Important decisions need to be made early in the development trajectory. Which functionality should be realized in software and which in hardware? What is the number and type of processors to be integrated? How should storage (both working memory and persistent disk storage) be organized? Is dedicated hardware development beneficial? How to distribute functionality? How to parallelize software? How can we meet timing, reliability and robustness requirements? The decisions should take into account the application requirements, cost and time-to-market constraints, as well as aspects like the need to re-use earlier designs or to integrate third-party components.

Industries typically adopt some form of model-based design for the software and hardware embedded in their systems. Fig. 1 illustrates such a process.

* This work was carried out as part of the Octopus project with Océ Technologies B.V. under the responsibility of the Embedded Systems Institute. This project was partially supported by the Netherlands Ministry of Economic Affairs under the Bsik program. This extended abstract is based on earlier papers describing the Octopus tool set [2] and our philosophy behind model-driven design-space exploration [7]; the full version of this extended abstract will appear as [3]. Nikola Trčka was employed by Eindhoven University of Technology when this work was performed.

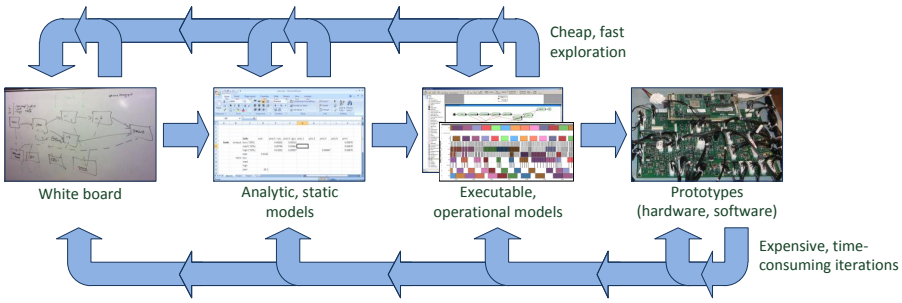

Fig. 1. Embedded-systems development is an iterative process typically involving several costly iterations over physical prototypes. Model-driven DSE avoids these iterations through fast and efficient exploration using models, improving time-to-market and leading to better controlled product quality.

Spreadsheets play an important role in early decision making about design alternatives. They provide a quick and easy method to explore alternatives at a high abstraction level. Executable operational models may then be developed for further exploration and/or synthesizing hardware and software. Current industrial practice uses such models mostly for the latter purpose, focussing on functional and structural aspects and not on extra-functional aspects such as timing and resource usage. Promising alternatives are realized in prototypes that include large parts of the software and hardware that are ultimately also used in the final system. Parts of the process may be iterated several times.

Design iterations through prototypes are time consuming and costly. Only a few alternatives can be explored. The number of design alternatives is extremely large though. The challenge is to more effectively exploit models for design-space exploration (DSE), avoiding design iterations over prototypes and extensive performance tuning and re-engineering at the implementation level. Spreadsheet analysis is suitable for a coarse pruning of options. However, it not well suited to capture system dynamics due to for example pipelined, parallel processing, data-dependent workload variations, resource scheduling and arbitration, variations in data granularity, etc. (see Fig. 2). High-level operational models can capture such dynamics. However, in industry, such models are not yet extensively used for DSE purposes. An important challenge is therefore to bridge the gap between spreadsheet type analysis and prototypes for DSE in industrial development practice. It is crucial to find the right abstractions, methods, and analysis techniques to support accurate and extensive DSE.

2 Model-Driven Design-Space Exploration

An important characteristic of DSE is that many different questions may need to be answered, related to system architecture and dimensioning, resource cost and performance of various design alternatives, identification of performance

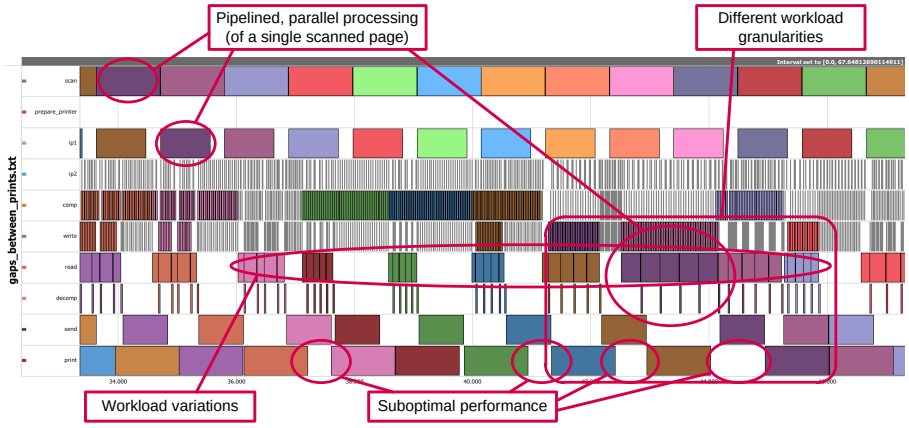

Fig. 2. An illustrative Gantt chart showing the execution of a printing pipeline. Dynamics in the processing pipeline cause hick-ups in print performance due to under-dimensioning of the embedded execution platform.

bottlenecks, sensitivity to workload variations or spec changes, energy efficiency, etc. Different models may be needed to address these questions. Models should be intuitive to develop for engineers from different disciplines (hardware, software, control), and they should be consistent with each other. Multiple tools may be needed to support the modelling and analysis. The Embedded Systems Institute (ESI) and its academic and industrial partners have picked up the challenge to develop systematic methods and tool support for DSE of software-intensive embedded systems. Given the characteristics of DSE, our approach is based on two important principles: separation of concerns and re-use and integration of existing techniques and tools (see Fig. 3).

ESI coordinates its efforts on model-driven DSE through the Octopus tool set (http://dse.esi.nl). The tool set architecture (Fig. 3, left) separates the modelling of design alternatives, their analysis, the interpretation and diagnostics of analysis results, and the exploration of the space of alternatives. This separation is obtained by introducing an intermediate representation, the DSE Intermediate Representation DSEIR, and automatic model transformations to and from this representation. This set-up allows the use of a flexible combination of models and tools. It supports domain-specific modelling in combination with generic analysis tools. Multiple analyses can be applied on the same model, guaranteeing model consistency among these analyses; different analysis types and analyses based on multiple models can be integrated in a single search of the design space. Results can be interpreted in a unified diagnostics framework.

The modelling in Octopus follows the Y-chart paradigm of [1, 5] (Fig. 3, right) that separates the concerns of modelling the application functionality, the embedded platform, and the mapping of application functionality onto the platform. This separation allows to easily explore variations in some of these aspects, for example the platform configuration or the resource arbitration, while

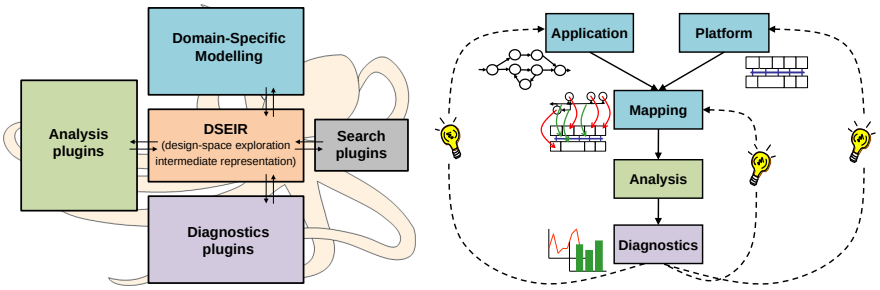

Fig. 3. Separation of concerns. The Octopus tool set architecture separates modeling, analysis, search and diagnostics through the intermediate representation DSEIR (left). Modeling and design-space exploration follow the Y-chart paradigm ([1, 5]; figure from [2]), separating application, platform and mapping aspects (right).

fixing other aspects, such as the parallelized task structure of the application. It also facilitates re-use of aspect models over different designs.

Intermediate representation DSEIR [7] plays a crucial role in the Octopus approach. It follows the Y-chart paradigm and is specifically designed for the purpose of model-driven DSE. DSEIR is realized as a Java library. The current implementation supports four views: application, platform, mapping, and experiment. DSEIR can be used through a Java interface, an Eclipse-based XML interface, and an Eclipse-based prototype GUI.

Applications in DSEIR are modelled as task graphs. Tasks communicate via ports, and their work loads are specified in terms of required services (such as CPU computation, storage needs). The use of services avoids direct references to the platform definitions, thus realizing the Y-chart separation of concerns. A platform consists of a number of resource declarations. Each resource has a capacity and provides services at a certain speed. The combination of service speed of a resource and service work load of a task can be used to compute task execution times. The mapping connects an application to a platform. It consists of resource allocations and priority specifications. Resource allocations specify whether or not preemption is allowed. Execution follows the dynamic priority scheduling paradigm. DSEIR models are all parameterized, with for example processor speeds, memory sizes, priorities, etc. as parameters. DSEIR has a well-defined operational semantics. The abstraction level is such that it is possible to efficiently and effectively capture the system behaviour and dynamics that is essential for a fast but accurate assessment of design alternatives. The experiment view of DSEIR allows to define sets of DSE experiments for selected models and model parameter settings. The analyses to be performed and the way to handle the output of the experiments can be specified as well.

The current tool set implementation supports modelling directly in DSEIR [7] as well as modeling of printer data paths for professional printers through a Domain-Specific Language (DSL) called DPML, the Data-Path Modelling Language [6]. Several types of analysis are supported. Performance analysis through

discrete-event simulation is supported via CPN Tools (http://cpntools.org), model checking is supported via Uppaal (http://www.uppaal.org), and dataflow analysis through SDF3 (http://www.es.ele.tue.nl/sdf3). Exploration support is available through JGAP (http://jgap.sourceforge.net) and the tool set has built-in support for dividing multiple analyses over the available processors of a multi-processor machine. Diagnostic support is provided through Excel, Gantt charts (see Fig. 2) and Pareto analysis [4].

3 Industrial Experiences

We have used Octopus in several case studies at Océ Technologies. These case studies involve design-space exploration of printer data paths of professional printers. Professional printing systems perform a variety of image processing functions on digital documents that support the standard scanning, copying and printing use cases, as well as many combinations and variations of these use cases. The printer data path encompasses the complete trajectory of the image data from source (for example the scanner or the network) to target (the imaging unit). The case studies show that the Octopus tool set can successfully deal with several modeling challenges, like various and mixed abstraction levels (from pages to pixels and everything in between), preemptive and non-preemptive scheduling, stochastic behavior, dynamic memory management, page caching policies, heterogeneous processing platforms with CPUs, GPUs, and FPGAs, realistic PCI and USB arbitration, etc. Our analyses identified performance bounds for printing pipelines and resource bottlenecks limiting performance, they gave designers a better understanding of the systems, confirmed design decisions (scheduling, arbitration, and caching), and suggested small design improvements (buffering, synchronization). Both DPML and DSEIR models can be made with little effort, very similar to the time investment needed for a spreadsheet model. An important advantage is that one model suffices to use different analysis tools. The analysis time in CPN Tools and Uppaal using models automatically generated from DSEIR models was always at least as good as with handcrafted models.

4 Challenges

The first experiences with the Octopus approach to model-driven DSE have been successful, but many challenges remain, both scientific challenges and challenges related to industrial adoption of model-driven DSE:

- How do we properly handle combinations of discrete, continuous, and probabilistic aspects? Such combinations materialize from combinations of timing aspects, user interactions, discrete objects being manipulated, physical processes being controlled, failures, wireless communication, etc.
- How can we effectively combine the strengths of different types of analysis, involving for example model checking, simulation, dataflow analysis, constraint programming, SAT/SMT solving, etc.? No single analysis technique

is suitable for all purposes. Integration needs to be achieved without resorting to one big unified model.
- How do we achieve scalability? Can we support modular analysis and compositional reasoning across analysis techniques, across abstraction levels, and for combinations of discrete, continuous, and probabilistic aspects?
- How to cope with uncertain and incomplete information? Information is often unavailable early in the design process, environment parameters and user interactions may be uncontrollable and unpredictable. How do we guarantee robustness of the end result of DSE against (small) variations in parameter values? Can we develop appropriate sensitivity analysis techniques?
- How do we cope with the increasing dynamics in modern embedded systems? Today's systems are open, connected, and adaptive in order to enrich their functionality, enlarge their working range and extend their life time, to reduce cost, and to improve quality under uncertain and changing circumstances. System-level control loops play an increasingly important role. What is the best way to co-design control and embedded hardware and software?
- Can we develop systematic, semi-automatic DSE methods that can cope with the complexity of next generations of high-tech systems? How do we provide model consistency when combining multiple models and analysis techniques?
- How do we handle the many different use cases that a typical embedded platform needs to support? How to support trade-off analysis over the many objectives that play a role in DSE?
- How do we incorporate DSE in the system development processes? This involves aspects like model calibration, model validation, and model versioning, but also linking DSE to code generation, hardware synthesis, and possibly model-based testing.
- How do we achieve industrial acceptance? Industrially mature DSE tools are a prerequisite. DSL support, tool chain customization, integration with other development tools, and training all need to be taken care of.

References

1. F. Balarin et al.: *Hardware-Software Co-design of Embedded Systems: The POLIS Approach*. Kluwer, 1997.
2. T. Basten et al.: Model-Driven Design-Space Exploration for Embedded Systems: The Octopus Toolset. Proc. ISoLA 2010, LNCS 6415, pp. 90–105. Springer, 2010.
3. T. Basten et al.: Model-Driven Design-Space Exploration for Software-Intensive Embedded Systems. In: T. Basten et al. (eds), Model-based Design of Adaptive Embedded Systems. Springer, 2013. To appear.
4. M. Hendriks et al.: Pareto Analysis with Uncertainty. Proc. EUC 2011, pp. 189–196. IEEE CS Press, 2011.
5. B. Kienhuis et al.: An Approach for Quantitative Analysis of Application-specific Dataflow Architectures. Proc. ASAP 1997, pp. 338–349. IEEE, 1997.
6. E. Teeselink et al.: A Visual Language for Modeling and Analyzing Printer Data Path Architectures. Proc. ITSLE 2011, 20 pp., http://planet-sl.org/itsle2011/.
7. N. Trcka et al.: Integrated Model-Driven Design-Space Exploration for Embedded Systems. Proc. IC-SAMOS 2011, pp. 339–346. IEEE CS Press, 2011.