Integrated Model-Driven Design-Space Exploration for Embedded Systems*

Nikola Trčka[†¶], Martijn Hendriks[‡], Twan Basten[§†], Marc Geilen[†], and Lou Somers[**¶]
[†] Electrical Engineering Department, Eindhoven University of Technology, The Netherlands
[¶] Department of Mathematics and Computer Science, Eindhoven University of Technology, The Netherlands
[‡] Computer Science Department, Radboud University Nijmegen, The Netherlands
[§] Embedded Systems Institute, The Netherlands
[**] Océ Technologies, The Netherlands
Email: n.trcka@tue.nl,m.hendriks@cs.ru.nl, a.a.basten@tue.nl, m.c.w.geilen@tue.nl, lou.somers@oce.com

***Abstract*—Embedded systems and their design trajectories are becoming increasingly complex, and there is a growing demand for performance, reliability, energy efficiency and low cost. To cope with these challenges, decision making early in the development trajectory needs to be supported by appropriate modeling and analysis. To achieve this support, we need to find the modeling abstractions that allow extensive design-space exploration, tune these modeling abstractions towards the users, and integrate support for different types of modeling and analysis.**

I. Introduction

The complexity of today's embedded systems and their development trajectories require a systematic, model-driven design approach, supported by tooling wherever possible. The biggest challenge are the many implementation possibilities that need to be considered, typically involving multiple metrics of interest (timing, resource utilization, energy usage, cost, etc.) and multiple design parameters (e.g., amount and type of processing units, memory size and organization, interconnect, scheduling and arbitration policies). The relation between design choices on the one hand and metrics of interest on the other is often very difficult to establish.

Most industries currently adopt some form of model-based design approach (see Fig. 1). A *spreadsheet* type of analysis is usually applied first, being a quick and easy method to produce feasible design alternatives. Each obtained solution is then verified (and fine-tuned) at the level of an *implementation model* which typically contains full functionality and real code. The spreadsheet analysis is typically pessimistic, leading to a possible over-dimensioning of the design that can only be corrected later in the process. The implementation models, on the other hand, are too costly to develop and too detailed to allow for any serious design-space exploration. There is, however, seldom any modeling and analysis applied at a level of abstraction in between these two extremes. A first important challenge in embedded system design is thus to find the right levels of abstraction in which extensive design-space exploration (DSE) is possible, while the accuracy of results (or trends) is sufficiently good.

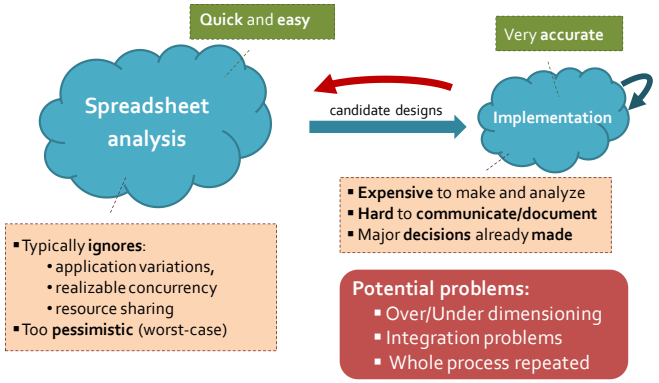

Fig. 1. The most common design practice: spreadsheet + implementation

Industry often still relies on spreadsheet analysis for DSE. There are several reasons for this. First, the need is felt to make design decisions quickly and without much (modeling) effort. Second, modeling and analysis tools in use in industry often require low-level details that are typically not known to the engineer in the early design stages (think of a third-party board-based design where certain aspects of the board are not known while the tool in use requires this board to be specified from the offered set of library components). Finally, there can be a mismatch between the in-house vocabulary of design engineers and the input language of the modeling tools, which may lead to frustration, incorrect models and inconsistent documentation. A second important challenge in embedded system design is how to support modeling abstractions that are closer to the user's perspective, and how to develop design tools that are highly customizable and adaptable, so that any specific problem at hand can be modeled, analyzed and documented with ease.

While performance and energy usage are the key metrics in DSE for embedded systems, systems are often required to satisfy other properties as well. Typical examples are absence of deadlocks and livelocks, feasibility of scheduling, hard real-time constraints, reliability guarantees, etc. These properties

*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 is partially supported by the Netherlands Ministry of Economic Affairs under the Bsik program.

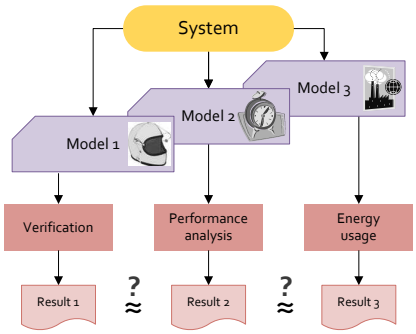

Fig. 2. Double modeling work and questionable consistency of results

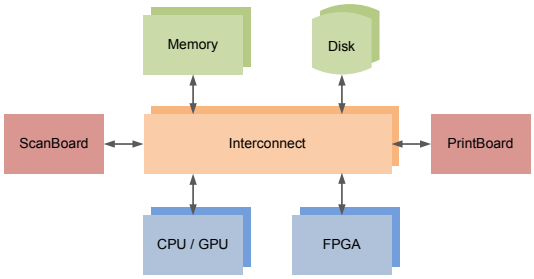

Fig. 3. A template for a typical printer data path architecture

are usually verified only after the exploration process, using different models that very often describe the same parts of the system. This is not only double work, but it might also lead to inconsistency and to designs that are rejected too late (see Fig. 2). No existing analysis tool is fit to cope with all the challenges of modern embedded system design, implying that more than one tool must be used. This leads to additional costs and possible consistency and integration problems. A third important challenge that we see is therefore how to facilitate the reuse of models across multiple tools, while ensuring consistency and proper integration.

In this paper, we elaborate on the challenges that we see in DSE for embedded systems. We propose an approach and toolset architecture that address the identified challenges, in the form of an integrated framework for model-driven DSE (MDDSE) for embedded systems. We survey the current state of the art in modeling, analysis and DSE in terms of this framework. We also show how the approach is made concrete in our Octopus toolset. We use an illustrative DSE problem from the professional printing domain to illustrate the challenges and to motivate our solutions.

The paper is organized as follows. Section II introduces the motivating DSE problem from the printing domain. Section III proposes the framework architecture to address the identified challenges, and surveys the state of the art. In Section IV, we introduce the Octopus toolset and summarize results we obtained in several case studies from the professional printing domain. The last section concludes.

II. Printer data path design

This section illustrates the typical DSE problem. We discuss the challenges we faced while conducting several case studies at Océ Technologies, involving the design of digital data paths in professional printers. These challenges are characteristic for other application domains as well.

Multifunctional printing systems perform a variety of image processing functions on digital documents that support the standard scanning, copying and printing jobs. The digital data path encompasses the complete trajectory of the image data from source (for example the scanner or the network) to target (the imaging unit).

Data path platform template: Fig. 3 shows a generic template of a typical embedded system architecture used for a digital data path of a professional printer. Several special-purpose boards are used to perform dedicated tasks, typically directly related to the actual scanning and printing. For computation, the data path platform may provide both general-purpose processors (CPUs, GPUs) and special-purpose FPGA-based units. RAM memory and hard disks are used for short- and long-term storage respectively. All the components are connected by interconnect infrastructure (e.g., PCI, USB). The architecture template focuses on the components needed for the digital image processing, leaving out user controls, network interfaces, etc. Note that the template is in fact generic for almost any embedded system.

Printer use cases: Each printer has dozens of simple use cases. The standard ones are scanning, printing, copying, scan-to-email and print-from-disk. Each use case typically involves several image processing steps such as rendering, zooming, rotation, compressing, half-toning, etc.; all these steps need to, or can, use different components in the platform. Moreover, print and scan use cases can be mixed. They can also be instantiated for different documents with different paper sizes, numbers and types of pages, etc. It is clear that this results in an explosion of possibilities.

We only explain the copying use case in more detail; the interpretation of the remaining basic use cases is similar. After scanning, each page is first processed by the ScanBoard and then further processed in several image processing steps; the resulting image is then printed by the PrintBoard. The image processing steps need to be done on some computation resource. Intermediate results are either stored temporarily in memory or permanently on disk. The latter is done for example to allow more than one copy of the document to be printed or to cope with errors. Uploading a processed image to disk can be done in parallel with any further processing.

Questions in data path design: The ScanBoard and PrintBoard are typically the first components to be selected for a specific printer. They determine the maximal possible scan and print speeds in pages per minute. The rest of the data path should be designed in such a way that these scan and print speeds are realized at minimal cost. Typically, a number of questions need to be answered early in the development trajectory. Which types of processing units should be used? How many? What clock speeds are needed? What amount of memory is needed? Which, and how many, busses are required? How should image processing steps be mapped onto

the resources? Other questions relate to scheduling, resource allocation and arbitration. What should be the scheduling and arbitration policies on shared resources? What are the appropriate task priorities? Can we apply page caching to improve the performance? How to allocate memory in RAM? How to share memory? How to minimize buffering in an FPGA? Is the memory allocation policy free of deadlocks? The data path design should take into account all basic use cases, as well as combinations such as simultaneous printing and scanning. It should also take into account different job types (text, images), paper sizes, etc. The design should be such that no bottlenecks are created for the most important use cases (normal printing, scanning, copying, on the default paper size for normal jobs). Performance can be traded off against costs for use cases that occur less frequently though. DSE should also provide insight in these trade offs. Furthermore, printing products typically evolve over time. This raises questions such as what is the impact of a new scan- or print board with higher specs on an existing data path design. It is clear that the complexity of DSE is huge.

Expressivity issues: Modeling the above uses cases for the sketched DSE is possible with a spreadsheet at a high level of abstraction as a first-order approximation. There are several risks involved though in doing so, that might lead to inaccurate results. First, there are various sources of variability. The complexity of a print job, the size of a compressed page and the execution time on a general purpose processor are all stochastic and rarely exactly predictable. Second, the flow of information is often too complex (and conditional) to be realistically captured by a static model like a spreadsheet. Think, e.g., about a smart-storage heuristic that takes a page from a disk only if it not still in memory. Third, pipelined and parallel steps in a job and simultaneously active print and scan jobs may interact on shared resources such as memory, busses and shared processors. Such dynamic interaction is difficult to capture in a static model. Finally, scheduling and arbitration policies that are often crucial for performance are hard if not impossible to model in a spreadsheet-type model.

Mixed abstraction levels: Although many image processing steps work on pixels or lines, most of the use cases can be accurately modeled at the page level. The throughput of the data path in images per minute is also the most important metric of interest. There are, however, cases where a mixture of abstraction levels is needed. FPGAs for example come with limited memory sizes. Only a limited number of lines of a page fit in FPGA memory. The page level thus becomes too coarse and a finer granularity of modeling is needed. Modeling complete use cases at the line or pixel level would make most analyses intractable though, so appropriate transitions between abstraction levels are needed.

The variety in analysis questions: The typical analysis questions in the list of DSE questions above might, in theory, all potentially be answered by a generic modeling and analysis tool; it is clear, however, that there is a variety of DSE questions of a very different nature. Deadlock and schedulability checks, for example, are best done using a model checker.

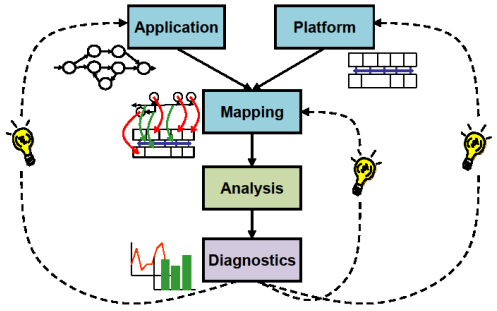

Fig. 5. The Y-chart DSE method

Any serious performance analysis would probably have to rely on simulation. Low-level FPGA buffer optimization can be done with fast, yet restrictive, dataflow analysis. This variety in analysis questions would suggest the use of different tools. This does require the development of multiple models though, with a risk of model inconsistencies and interpretation difficulties. Ideally, one unifying model would form a basis for analyses performed with different techniques and tools.

Customization for the printer domain: Principles of printer platforms and printer use cases are not rapidly changing. Having the models written in a printer-specific language, and maintaining a library of those models, would drastically decrease the modeling effort, reduce modeling errors, and improve communication and documentation of design choices.

III. Integrated High-Level Model-Driven Design-Space Exploration

This section motivates what in our view is an ideal architecture for an MDDSE framework. We identify some key ingredients of such an architecture, as answers to the challenges raised in the introduction and the previous section. Along the way, we survey existing methods, languages, and tools that fit in such a framework. Fig. 4 shows the end result; the remainder of this section elaborates on this figure.

A. Separation of concerns: Y-chart based design

The first step in achieving the right level of (modeling) abstraction is a clear separation of concerns. We propose to follow the Y-chart philosophy [2], [24], which is popular in hardware design. This philosophy is based on the observation that DSE typically involves the co-development of an application, a platform, and the mapping of the application onto the platform (see Fig. 5). Diagnostic information is used to, automatically or manually, improve application, platform, and/or mapping. This separation of application, platform, and mapping is important to allow independent evaluation of various alternatives of one of these system aspects while fixing the others. Often, for example, various platform and mapping options are investigated for a fixed (set of) application(s).

The Y-chart philosophy illustrates the key activities involved in DSE: modeling of design alternatives, analyzing metrics of interest, diagnosing potential points of improvement, and exploring the space of design alternatives based on the diagnosis. The framework of Fig. 4 follows this breakdown in activities.

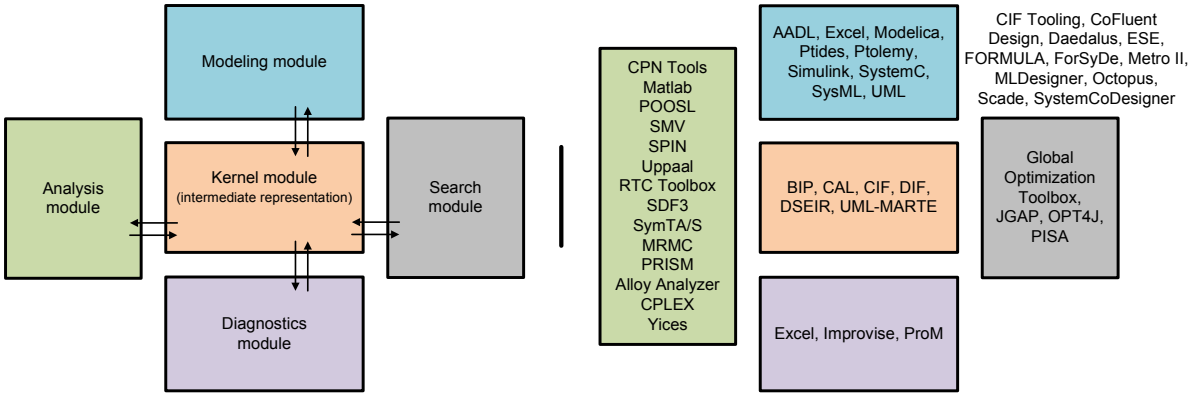

Fig. 4. Conceptual architecture of an integrated framework for MDDSE (left); methods, languages and tools that fit within this conceptual framework (right)

B. Application-centric domain-specific modeling

As mentioned in the introduction, spreadsheet analysis is typically only a coarse approximation of system behavior. It does not fully take into account dynamics such as realizable concurrency, variations in application behavior, and resource behavior and sharing. Implementation models on the other hand require that these aspects are modeled in full detail. For MDDSE, given the complexity of the DSE process, an intermediate modeling abstraction level is needed, that is simple and allows efficient and accurate analysis.

Another important aspect identified before is that modeling abstractions need to appeal to the user and adhere to his or her intuition. Domain-specific abstractions and user-customization should therefore preferably be supported. We furthermore propose application-centric modeling. Dynamic behavior in the system is determined by the application part. The applications should therefore be leading, and their models should capture all behavior variation and all concurrency explicitly. We propose to model platforms as sets of resources that have no behavior of their own; their purpose is only to (further) restrict application behavior and to introduce proper timing. This leads to simple, predictable, and tractable models. Scheduling and mapping can then be unified into the concept of prioritized dynamic binding. If needed, modeling of complex resource behavior (work division, run-time reconfiguration, etc.) can always be supported through (automatic) translations into application behavior.

A variety of modeling environments and approaches in use today, either in industry or in academia, can support the envisioned modeling style. We mention some of them, without claiming to be complete: AADL [1], Modelica [28], Ptides [8], Ptolemy [16], Simulink [33], SystemC [37] and UML [40].

C. High-level model-driven DSE

The Y-chart separation of concerns in combination with application-centric domain-specific modeling supports a design approach that can be seen as a Y-chart oriented refinement of the spreadsheet approach; only the necessary dynamic aspects are added, in sufficient detail, to achieve a one-to-one correspondence with the user's understanding of the system.

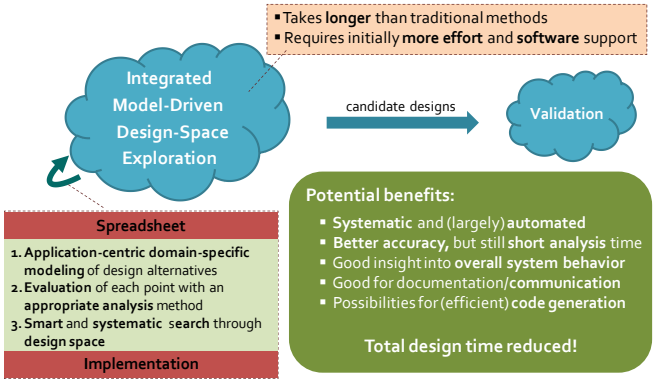

Fig. 6. High-level model-driven design-space exploration

Accurate DSE can then be done in an early design phase, replacing the spreadsheet. The abstraction level is such that large spaces of design alternatives can be explored. Different points in the space can be analyzed to address the typical DSE questions as outlined in the previous section. The risk of finding problems at the implementation level thus reduces. In addition, the proposed model-driven approach may facilitate (semi-)automatic generation of final implementations, ease communication and documentation, and enable model reuse. Fig. 6 illustrates what in our view is the suitable modeling and design approach for DSE, also showing its benefits.

D. Analysis, Search, Diagnostics

In Section II, we have seen an illustration of the variety of design questions that are typically investigated early during development. No single tool or analysis method is fit to address all these questions. We foresee the combined use of different analysis tools in one DSE process. A wide variety of, mostly academic, but also some commercial, tools is available that can be used to support DSE.

For quick exploration and performance optimization, discrete-event simulators such as CPN Tools [17], MatLab [27], and POOSL [39] are suitable. Model checkers such as SMV [34], SPIN [18], or Uppaal [5] can be used for functional verification, protocol checking, and schedule and timing optimization. Model checkers may not be able to cope

with the full complexity of modern embedded systems, but they may play an important role in verifying and optimizing critical parts of the system. Yet other tools, such as the RTC Toolbox [31], SDF3 [35], or SymTA/S [36], are suited for timing analysis of data-intensive system parts, such as image processing chains.

Questions regarding performance, reliability, and schedulability under soft deadlines can be answered by increasingly popular probabilistic model checking techniques, using tools like PRISM [29] and MRMC [22]. These techniques enhance the expressivity of regular model checking, allowing for more realistic modeling of aspects such as arrival rates and failure times.

In recent years, also constraint programming and SAT/SMT solvers have gained popularity. With the arise of more powerful computers and improvements in the techniques themselves, tools like CPLEX [7], Alloy Analyzer [20], and Yices [44] are increasingly often applicable to find optimal or feasible solutions for system aspects such as resource bindings or schedules.

Besides support for the analysis of metrics for design alternatives or for the optimization of parts of the system, we also need support to explore the huge space of design alternatives. The MathWorks Global Optimization Toolbox [26] supports a wide variety of customizable search algorithms. The JGAP library [21] is a Java library for developing genetic search algorithms. OPT4J [25] and PISA [6] are customizable genetic search frameworks that support DSE.

Most tools above already give very good diagnostic reports, which in many cases can be successfully converted and interpreted in the original domain. Visualization of analysis results is also a very useful diagnostic method. Excel is useful in this context, and sophisticated visualization/mining can be done using e.g. Improvise [19] or ProM [30].

E. Intermediate representation: flexibility, consistency, customization

It cannot be expected that designers master the wide variety of modeling languages and tools mentioned so far, and apply them in combination in DSE. To successfully deal with integration, customization and adaptation of models, as well as to facilitate the reuse of models across tools and ensure consistency between models, we foresee the need for some form of intermediate representation (IR) to connect different languages and tools in a DSE process. Such an IR must in the first place be able to model the three main ingredients of the Y-chart in an explicit form. It should not have too many specific constructs to facilitate translation from different domain-specific modeling languages and to different target analysis tools, yet it must be powerful and expressive enough to accommodate developers. A good balance between modeling expressiveness and language complexity is needed. Besides the Y-chart parts, the IR must provide generic means to specify design alternatives, quantitative and qualitative properties, diagnostic information, etc., i.e., all ingredients of a DSE process. In our view, the IR should be formally specified, to avoid interpretation problems and ambiguity between different models and analysis results. The IR does not need execution support though, because execution can be done through back-end analysis tools. Intermediate representations and languages like BIP [4], CAL [10], CIF [42], DSEIR [3], DIF [13] and UML-MARTE [41] are examples of languages that could be adapted to fully support MDDSE as sketched in this section.

F. Integrated service-based software architecture

To realize the vision outlined in this section, we propose a service-based implementation of the framework of Fig. 4. Modules should communicate with other modules through clean service interfaces. Domain-specific modeling tools with import/export facilities to the IR are in the modeling module. The analysis module provides analysis services such as performance evaluation, formal verification, schedulability analysis, etc. The diagnostics module provides ways to visualize analysis results and gives high-level interpretations. The search module contains support for search techniques to be used during DSE. The kernel module coordinates the information flow between the other modules and implements the IR.

G. DSE frameworks

There are already several frameworks that support DSE, or aspects of it, for various application domains. Examples are CoFluent Design [9], Daedalus [15], ESE [43], FORMULA [14], ForSyDe [32], METRO II [12], MLDesigner [38], Octopus [3], SCADE [11], and SystemCoDesigner [23]. From these frameworks, METRO II and our own framework Octopus are closest to the views outlined in this section. The large number of already available languages, tools, and frameworks are a clear indication of the potential of high-level modeling, analysis, and DSE. We started the development of Octopus to realize and evaluate our ideas on MDDSE. Octopus explicitly aims to leverage the combined strengths of existing tools and methods in DSE. The next section describes the latest version of Octopus, its IR, and our experience with DSE in the professional printer domain. Compared to the presentation in [3], both the toolset itself and the IR have been extended and adapted, to better fit with the views outlined in this paper.

IV. The Octopus Toolset

The Octopus toolset (its kernel module to be precise) is written in Java and Fig. 7 shows its current architecture. The central notion is the IR called DSEIR (Design-Space Exploration Intermediate Representation), explained in detail below. Two front-ends are used as editors, a generic Eclipse-based DSEIR editor and a printer-specific editor. At present, only genetic programming is used in the search module, through the JGAP library. Several types of analysis are supported: performance analysis by simulation (using CPN Tools), model checking (using Uppaal), best/worst latency calculation and schedulability analysis (Uppaal, through an extended translation involving task deadlines), and worst-case throughput calculation (using SDF3). All transformations from and to DSEIR are fully automated. The mentioned tools are

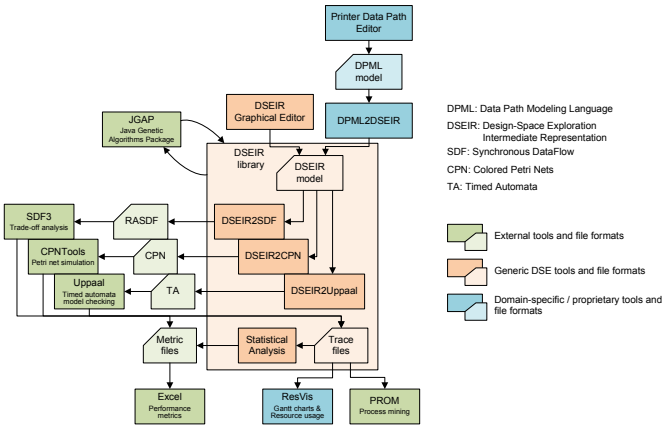

Fig. 7. The current implementation architecture of the Octopus toolset

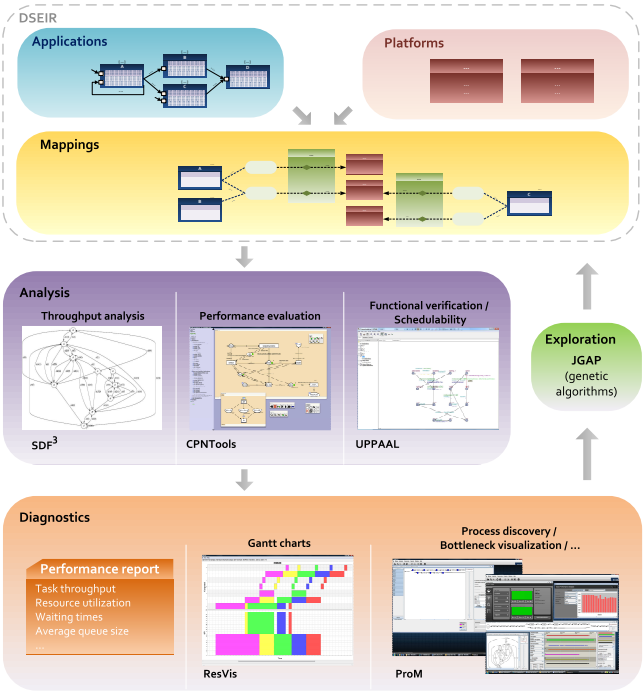

Fig. 8. The Octopus toolset – Implementing the Y-chart

used in their batch modes, requiring no user intervention. The diagnostics module contains Gantt-chart based visualizations of execution traces (through a proprietary tool ResVis), an Excel output, and several mining features through the link with ProM. Fig. 8 shows the Octopus toolset from the Y-chart perspective, giving also a hint on what DSEIR looks like.

A. DSEIR

DSEIR is a Y-chart based modeling language, designed following the ideas put forward in Section III. We explain the main concepts of DSEIR using a small printer use case.

1) Services: The concept of a service is the only concept that is shared between all the three parts of the Y-chart. Applications need services, which are in turn provided by the platform. Typical services are the computation, storage, and transfer services. A unit of load is assumed for each service.

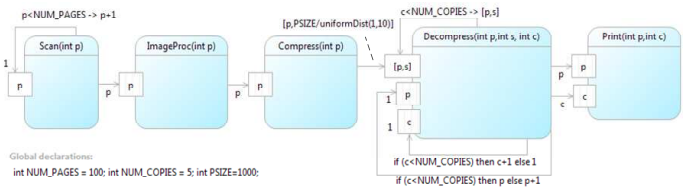

Fig. 9. DSEIR application graph of a printer use case

2) Application: An application in DSEIR consists of global variable declarations and of a task-flow graph. Global variables store data that needs to be visible to every element of the application; the task-flow graph specifies the flow of control and of data between individual tasks.

Fig. 9 shows the application model of a small printer use case, drawn in the DSEIR graphical editor, where each page coming from the scanner is processed, compressed and implicitly stored to a disk, and then taken from the disk for decompression and multi-copy printing. Communication between ImageProc and Compress goes via memory; the disk transfer between Compress and Decompress is ignored.

The printer application defines three constants as global variables, for the number of pages in the document, the number of copies to be printed for each page, and the original page size (considered the same for every page).

Tasks: A task models an operation that is considered atomic in the current level of abstraction (like line-scanning or page-printing). It can be seen as a function, having zero or more typed parameters. When each parameter of a task is given a value, a task instance is obtained, which is similar to a function call. This instance can fire (execute), potentially producing a result that is sent to another (or the same) task, or modifying global variables. The Scan task from Fig. 9 has only one parameter p, indicating the page number. Task Decompress has three parameters, indicating the page number, the size of the page and the current copy instance.

Ports and bindings: A task contains zero or more ports that can store data received from other tasks and that specify initial values. These data are represented in the form of tokens carrying values. Ports are unbounded and typed. The tokens (or their values to be precise) are used to bind task parameters to values, forming executable task instances. When a tasks instance fires, the corresponding tokens are removed.

Each port has a binding expression associated to it (drawn inside the port). A binding is formed if each parameter is given a unique value when every binding expression is set equal to the value of some token in its port. In this way, we generalize the notion of assignment of values to task parameters. If the port is unordered, tokens are non-deterministically considered for bindings; if it is FIFO, the first token must be used.

In the model from Fig. 9 all ports are integer ports except the first port of Decompress that can store integer arrays of size two. The port of Scan has one token initially, with value 1 (the number of the first page); the second and the third port of Decompress have the same initial values that indicate the

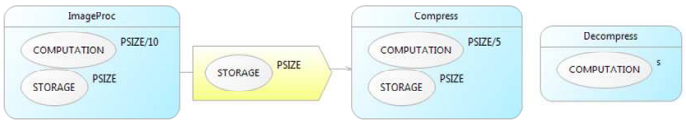

Fig. 10. Task load and handover

Fig. 11. A simplified printer platform

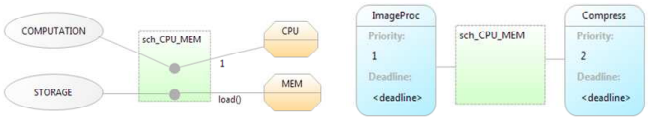

Fig. 12. Schedulers and mapping in DSEIR

first page and the first copy number. All binding expressions are simple task parameters, except again for the first port of Decompress where two parameters are combined into one array. This port is also the only unordered port in the model as its tokens are reused between firings. Note that in Decompress, the same parameter p is used in two ports, to make sure that the pages are always processed in order.

Edges: Edges connect tasks to ports. Every edge has an expression, representing the result that the source task is sending to the destination port, and a condition, a Boolean expression (considered true if absent) specifying when this result should be sent and when it should simply be ignored.

Most edges in Fig. 9 simply carry the page and the copy number over to the next task. The self-loop in the scan task is conditional; it produces tokens incrementally until it reaches the value NUM_PAGES. Compress reduces the size of each page randomly, by a factor uniformly distributed between 1 and 10. This is made explicit in the edge going from Compress to Decompress. The two bottom edges of Decompress make sure that the parameters p and c are properly incremented (or reset). The upper edge of the same task is to make sure that the page token is not reproduced if the last copy has been processed.

Task load and handover: To elegantly define resource requirements without mixing application and platform concepts, every task specifies the load it imposes on specific services. This load is evaluated for each instance of this task, and can depend on the values of its parameters. At present, it must be manually specified.

To model resource reservations and buffering, DSEIR introduces the notion of handover. A handover is an expression associated to each edge and service, representing that the rights over a certain amount of the resource executing this service shall be transferred to the destination task when the source task is completed.

Fig. 10 shows the load and the handover perspective of three of the printer tasks. The picture shows that Decompress requires a computation service; it imposes a load to this service that is equal to the (compressed) page size s it is currently processing. ImageProc and Compress also require computation that is proportional to the uniform, uncompressed page size. As they communicate via a memory buffer, they also impose a load on storage. The figure in between the two tasks indicate that image processing will hand over PSIZE amount of the resource it holds for storage to Compress. Note that the specific resource is not known at this point.

3) Platform: A platform in DSEIR is represented as a set of resources. The cooperation between resources is not explicitly modeled, but inherited from the application model after the mapping. There is no distinction between a computation, storage, or communication resource.

A resource is an entity that, besides a name, has 1) a total capacity, denoting the total amount of this resource that can be occupied; and 2) a speed, described as the time it takes this resource to process one unit of load, given a service it will perform and its current allocation to a task. A resource can provide more than one service, and more than one resource can provide the same service, typically with a different speed/capacity.

Fig. 11 shows a part of the platform for the printer case. MEM is the only resource providing storage; its capacity is 100000 and it has infinite speed (load is processed in zero time). The CPU resource provides computation. It has capacity 1, indicating that it can only process one task at a time. The CPU can process 3 load units in one unit of time.

4) Mapping: Mappings in DSEIR are very generic, combining dynamic allocation, scheduling, and a priority mechanism. Tasks are mapped to (user-defined) schedulers. Two tasks can use the same scheduler, but one task cannot use more than one scheduler.

A scheduler controls one or more resources. For each service that a task mapped to it requires, the scheduler must specify the amount of a resource that would be allocated to perform this service. To offer full flexibility, this amount can depend both on the load of the task and on the run-time state of the resource and the scheduling queue.

Fig. 12 shows the scheduler to which the ImageProc and Compress tasks are mapped. This scheduler assigns the CPU to every task requiring computation, and it gives this task the amount of MEM that is equal to the load this task imposes on the storage service. A task instance running with a higher priority is considered for scheduling before a task instance running with a lower priority. Priorities can dynamically change and two instances can have the same priority. For the printer case, we give higher priority to tasks that are further in the pipeline (cf. Fig. 12).

5) Putting the system together: A user can specify several applications, platforms, and mappings in one single model. A full system specification is obtained by taking one instance form each of these perspectives. This model is converted to an internal Java format, from which point several predefined services, as mentioned in the introduction to this section, can be used to facilitate design space exploration.

B. Experiences in the professional printing domain

We have used Octopus in several printer case studies at Océ Technologies. These case studies show that DSEIR 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, 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). DSEIR models can be made with little effort, very similar to the time investment needed for a spreadsheet model, and 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. The actual exploration of design alternatives was done mostly manually. It remains a challenge to cover the huge number of possible use cases.

V. Conclusions

The developments in modeling and analysis of embedded systems have come to the point that model-driven DSE is becoming a realistic option in industrial development processes. MDDSE has the potential to replace or complement the spreadsheet-type analysis typically done nowadays. It supports the systematic evaluation of design choices early in the development. MDDSE can thus reduce the number of design iterations, improve product quality, and reduce cost. To facilitate the practical use of MDDSE, we believe it is important to leverage the possibilities and combined strengths of the many existing languages and tools, and to present them to designers through domain-specific abstractions. We therefore set out to develop our MDDSE framework Octopus that intends to integrate languages and tools in a unifying framework. It aims to support the complete DSE process. It enables customization to specific application domains, allows model reuse among tools, and provides model consistency. An important remaining challenge is to bring together existing tools and analysis techniques in a systematic DSE method. We further plan to extend Octopus to also support automatic model abstraction and refinement, code generation, and synthesis.

References

[1] AADL. www.aadl.info, 2011.

[2] F. Balarin et al. *Hardware-Software Co-design of Embedded Systems: The POLIS Approach*. Kluwer, 1997.

[3] T. Basten et al. Model-Driven Design-Space Exploration for Embedded Systems: The Octopus Toolset. In *Proc. ISoLA 2010*, volume 6415 of *LNCS*, pages 90–105. Springer, 2010. http://dse.esi.nl.

[4] A. Basu, M. Bozga, and J. Sifakis. Modeling Heterogeneous Real-time Components in BIP. In *Proc. SEFM 2006*, pages 3–12. IEEE CS, 2006.

[5] G. Behrmann et al. Uppaal 4.0. In *Proc. QEST 2006*, pages 125–126. IEEE CS Press, 2006. www.uppaal.com.

[6] S. Bleuler et al. PISA – A Platform and Programming Language Independent Interface for Search Algorithms. In *Proc. EMO 2003*, LNCS 2632, pages 494–508. Springer, 2003. www.tik.ee.ethz.ch/pisa.

[7] IBM ILOG CPLEX Optimizer. www.ibm.com/CPLEX, 2011.

[8] P. Derler, E. A. Lee, and S. Matic. Simulation and Implementation of the PTIDES Programming Model. In *Proc. IEEE/ACM DS-RT 2008*, pages 330–333. IEEE CS Press, 2008.

[9] CoFluent Design. CoFluent Studio. www.cofluentdesign.com, 2011.

[10] J. Eker and J.W. Janneck. CAL Language Report Specification of the CAL Actor Language. Technical Report UCB/ERL M03/48, EECS Department, University of California, Berkeley, 2003.

[11] Esterel Technologies, Scade Suite. http://www.esterel-technologies.com/products/scade-suite, 2011.

[12] A. Davare et al. A Next-Generation Design Framework for Platform-based Design. In *DVCon 2007*, February 2007.

[13] C.-J. Hsu et al. DIF: An Interchange Format for Dataflow-Based Design Tools. In *Proc. SAMOS 2004,* LNCS 3133, pages 3–32. Springer, 2004.

[14] E.K. Jackson et al. Components, platforms and possibilities: towards generic automation for MDA. In *Proc. EMSOFT 2010*, pages 39–48. ACM, 2010.

[15] H. Nikolov et al. Daedalus: Toward Composable Multimedia MP-SoC Design. In *Proc. DAC 2008*, pages 574–579. ACM, 2008. http://daedalus.liacs.nl.

[16] J. Eker et al. Taming heterogeneity - the Ptolemy approach. *Proc. of the IEEE*, 91(1):127–144, 2003.

[17] K. Jensen et al. Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems. *STTT*, 9(3–4), 2007.

[18] G.J. Holzmann. *The SPIN Model Checker*. Addison-Wesley, 2003. http://spinroot.com.

[19] Improvise. www.cs.ou.edu/ weaver/improvise/index.html, 2011.

[20] D. Jackson. *Software Abstractions: Logic, language, and Analysis*. MIT Press, 2006.

[21] JAVA Genetic Algoritmhs Package. http://jgap.sourceforge.net, 2011.

[22] J.-P. Katoen et al. The Ins and Outs of the Probabilistic Model Checker MRMC. In *Proc. QEST 2009*, pages 167–176. IEEE CS Press, 2009.

[23] J. Keinert et al. SystemCoDesigner – An Automatic ESL Synthesis Approach by Design Space Exploration and Behavioral Synthesis for Streaming Applications. *ACM TODAES*, 14:Art. No. 1, 2009. http://www12.informatik.uni-erlangen.de/research/scd.

[24] B. Kienhuis et al. An Approach for Quantitative Analysis of Application-specific Dataflow Architectures. In *Proc. Application-Specific Systems, Architectures and Processors*, pages 338–349. IEEE, 1997.

[25] M. Lukasiewycz et al. Opt4J: Meta-heuristic Optimization Framework for Java. htpp://opt4j.sourceforge.net, 2010.

[26] MathWorks - global optimization toolbox. www.mathworks.com/products/global-optimization, 2011.

[27] MATLAB. www.mathworks.com/products/matlab, 2011.

[28] Modelica and the Modelica Association. www.modelica.org, 2011.

[29] PRISM. www.prismmodelchecker.org, 2011.

[30] ProM - Process mining workbench. www.promtools.org/prom6/, 2011.

[31] Modular Performance Analysis with Real-Time Calculus: RTCtoolbox. www.mpa.ethz.ch/Rtctoolbox, 2011.

[32] I. Sander and A. Jantsch. System Modeling and Transformational Design Refinement in ForSyDe. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(1):17–32, 2004.

[33] Simulink - Simulation and Model-Based Design. http://www.mathworks.com/products/simulink, 2011.

[34] The SMV system. www.cs.cmu.edu/ modelcheck/smv.html, 2011.

[35] S. Stuijk, M.C.W. Geilen, and T. Basten. SDF^3: SDF For Free. In *Proc. ACSD 2006*, pages 276–278. IEEE CS, 2006. www.es.ele.tue.nl/sdf3.

[36] Symtavision SymTA/S. www.symtavision.com/symtas.html, 2011.

[37] Open SystemC Initiative (OSCI). www.systemc.org, 2011.

[38] MLDesign Technologies. MLDesigner. www.mldesigner.com, 2011.

[39] B.D. Theelen et al. Software/Hardware Engineering with the Parallel Object-Oriented Specification Language. In *Proc. MEMOCODE 2007*, pages 139–148. IEEE CS Press, 2007.

[40] Object Management Group - UML. www.uml.org, 2011.

[41] The UML Profile for MARTE: Modeling and Analysis of Real-Time and Embedded Systems. www.omgmarte.org, 2011.

[42] D.A. van Beek et al. New Concepts in the Abstract Format of the Compositional Interchange Format. In *Proc. ADHS 2009*, pages 250–255. Elsevier, 2009.

[43] I. Viskic, L. Yu, and D. Gajski. Design Exploration and Automatic Generation of MPSoC Platform TLMs from Kahn Process Network Applications. In *Proc. LCTES 2010*, pages 77–84. ACM, 2010.

[44] Yices: An SMT Solver. http://yices.csl.sri.com, 2011.