Le paradigme « Y-Chart » : une discussion sur le développement dirigé par les modèles

Malgré sa grande complexité si on s'attarde aux détails, le processus de développement d'un logiciel ou d'un système embarqué peut être résumé par deux grandes étapes :

- comprendre et décrire le besoin (le quoi);
- implanter la solution (le comment).

La communauté du logiciel s'intéresse à ces deux étapes ainsi que le passage de l'un à l'autre depuis plusieurs années. Dans les années 70, des techniques structurées pour aborder ces sujets ont vu le jour afin de pallier aux nombreux problèmes reliés aux approches précédentes généralement très ad hoc [23]. Au fil des années, influencées par l'émergence de nouveaux paradigmes tel que l'orienté objet, les techniques structurées ont donné place aux techniques d'analyse et de conception orienté objet [12]. Malgré les énormes efforts investis pour avancer la science de l'élaboration des logiciels, le passage du « quoi » au « comment » reste toujours très difficile. Une cause principale de cette difficulté est que le passage du « quoi » au « comment » se traduit souvent par une intervention humaine qui fait l'élaboration d'artéfacts pour implémenter le « comment » basé sur sa compréhension du « quoi ». Ce type d'intervention est souvent long et sujet à l'erreur.

Au début des années 80, une nouvelle approche a été proposée pour rapprocher le « quoi » du « comment », la méthode d'analyse orienté objet et de conception récursive Shaler-Mellor [66]. Cette approche consiste à modéliser le « quoi » de manière assez précise qu'il soit possible d'utiliser une approche par traduction automatisée pour générer le « comment » final. Cette approche préconise aussi, basée sur le principal de l'orthogonalité des aspects, une modélisation séparée des aspects

purement logiciel des aspects de contraintes architecturales, ainsi que la spécification des « ponts » afin de faire le lien entre les deux aspects. Cette approche a été raffinée dans les années 90 et 2000 afin de donner lieux à la conception dirigée par les modèles [11] et l'architecture dirigée par les modèles [45]

Le paradigme « Y-Chart » développé en 1997 peut être clairement classifié comme une approche dirigée par les modèles au même titre que la méthode Shaler-Mellor Quoi que le travail original de [43] ne fait mention que de l'approche orienté objet et non le travail de [66], il est difficile de croire que l'approche n'a pas été influencée par ce dernier ou du moins par les principes véhiculés dans le milieu du logiciel grâce à celui-ci au début de années 90.

L'article qui suit présente le paradigme « Y-Chart », un exemple d'une méthodologie pour la conception de systèmes embarqués dirigé par les modèles. Une grande portion de l'article est consacrée à la présentation/discussion de diverses approches pour l'implémentation de cette méthodologie, un sujets qui n'ont pas beaucoup été traités dans la littérature. Cet article peut-être perçu comme une extension de l'état de l'art de cette thèse.

Les contributions principales de cet article sont:

- une présentation à caractère pédagogique de la méthodologie « Y-Chart », des modèles de calcul et des modèles d'architecture;
- une présentation de trois implémentations de la méthodologie « Y-Chart » afin de faire la comparaison de leurs choix de conceptions.

Les pages suivantes contiennent une copie de l'article [50], dans son format original (sauf la numérotation des pages), soumis à *ACM Transactions on Embedded Computing Systems*.

Y-Chart Based System Design: A Discussion on Approaches

JAMES LAPALME
University of Montréal, Department of Computer Science and Operations Research, Canada
and
BART THEELEN
Eindhoven University of Technology, Department of Electrical Engineering, Netherlands
and
NIKOLAY STOIMENOV
ETH Zürich, Computer Engineering and Networks Laboratory, Switzerland
and
JEROEN VOETEN
Eindhoven University of Technology, Department of Electrical Engineering; Embedded Systems Institute; Netherlands
and
LOTHAR THIELE
ETH Zürich, Computer Engineering and Networks Laboratory, Switzerland
and
EL MOSTAPHA ABOULHAMID
University of Montréal, Department of Computer Science and Operations Research; Canada

Embedded systems are a source of technology that facilitates our modern lifestyle. In order to do so, they tend to increase in complexity as well as integrate in are day-to-day activities. To meet the market's expectations on technological improvement, time-to-market objectives for introducing innovative embedded systems are shorter than ever. Over the last decade, model-based design has been a subject of great interest as a means to accelerate the design of embedded systems. The Y-chart paradigm is a principal approach to model-based embedded system design. Despite the simplicity and conciseness of this paradigm, it has been implemented in several different ways by various methodologies. This variety in implementation designs is due to the particular emphasis a methodology puts on the different steps of the paradigm (application modeling, platform modeling, mapping, analysis and synthesis). This article explores this variety by examining and comparing three Y-chart based design methodologies: Metropolis, the Distributed Operation Layer incorporating Modular Performance Analysis and the Y-chart variant of the Software/Hardware Engineering methodology. These methodologies have been chosen because they: cover a broad domain of applications, have been developed on a relatively long period of time and are representative of typical Y chart approaches. Moreover, these implementations of the paradigm present interesting design approaches which are worth comparing.

This article (i) presents the concepts underlying the Y-chart paradigm as well as models of computation and models of architecture, (ii) discusses the three mentioned implementations, and (iii) compares these implementations to highlight important design differences. The examination and comparison show that the Y-chart paradigm is a very flexible framework that can be implemented in many ways.

Categories and Subject Descriptors: C.0 [**Computer Systems Organization**]: Modeling of Computer Architecture, System Architectures, Systems Specification Methodology; F.1.1 [**Computation by Abstract Devices**]: Models of Computation; B.1.2 [**Performance and Reliability**]: Performance Analysis and Design Aids; I.6.4 [**Simulation and Modeling**]: Model Validation and Analysis; I.6.5 [**Simulation and Modeling**]: Model Development—*Modeling Methodologies*

General Terms: Design, Languages, Measurement, Performance, Verification

Additional Key Words and Phrases: Design-Space Exploration, Embedded Systems, Model of Architecture, Model of Computation, Y-Chart

---

1. INTRODUCTION

The decreasing time-to-market for modern embedded systems encourages the industry to strive for design reuse between multiple products. Moreover, the constantly increasing consumer demands for full featured systems requires the use of high-performance platforms. In order to cope, systems designers often consider platforms such as Multi-Processor Systems-on-Chip (MPSoC) because they allow reuse and offer high-performance. MPSoC include a combination of multiple processors and specialized processing elements to allow the execution of multiple applications in parallel. They also offer the flexibility to optimise aspects such as performance and energy consumption. When elaborating a system, designers face some crucial questions:

—*Which platform is most suitable for realizing the requested functionality?*

—*How to exploit the parallelism provided by the chosen platform efficiently?*

—*Is the application software parallelized in a suitable way?*

Figure 1 illustrates how the Y-chart paradigm introduced in [Kienhuis et al. 1997] provides a framework for answering these questions. This work should must not

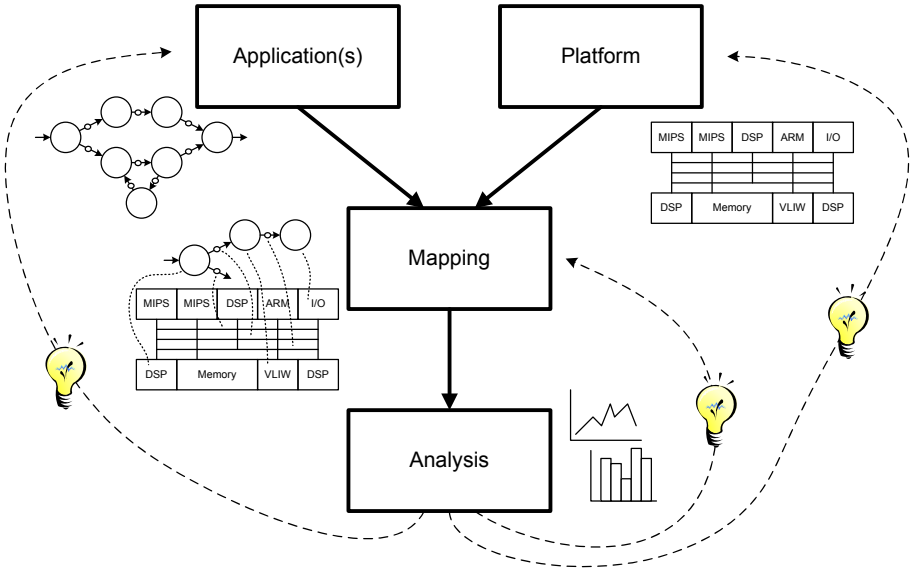


Fig. 1. Design-space exploration with the Y-chart paradigm

be confused with the Gajski and Kuhn Y-chart [Gajski and Kuhn 1983] which is a taxonomy for visualizing design views as well as design hierarchies. Modern embedded systems such as mobile phones support multiple *applications* such as MP3 players and Internet browsers. These applications are targeted to execute on a collection of processors (and specialized processing elements) that make up a *platform* [Martin 1998; H. Chang and Todd 1999; Sangiovanni-Vincentelli and Martin 2001]. By defining a *mapping* to specify which processor (or specialized processing elements) executes what functions of each application (allocation and binding) and at which moments in time (scheduling), one can predict the resulting performance of the overall solution design. The analysis of these results may give hints for improving the application software, the platform design and/or the mapping. The Y-chart paradigm envisions to iteratively apply such improvements until finding a solution that satisfies the end requirements. The final solution is then implemented to produce the desired product. The Y-chart paradigm was initially introduced by Kienhuis et al. [Kienhuis et al. 1997] in the context of dataflow-oriented systems. It was also independently applied in the POLIS project [Balarin 1997] for control dominated systems (e.g. automotive). Recently, it has also been used in others control-oriented domains [Baleani et al. 2000; Sgroi et al. 2001; Zeng et al. 2006]. The work introduced by Kienhuis consisted of Stream-Based Functions for describing applications, a C++ inspired Pamela library for specifying platforms, and a simulator called ORAS to perform the analysis part [Kienhuis 1999].

Although the key concept underlying the Y-chart paradigm of explicitly separating application descriptions from a platform specification is relatively simple, methodologies like POLIS, Metropolis [Balarin et al. 2003], SPADE [Lieverse et al. 2001], and Cadence's Virtual Component Co-Design Environment [Krolikoski et al. 1999] use very different implementations of the paradigm. This article discusses the

Y-chart paradigm for design-space exploration of distributed embedded systems by investigating alternative approaches to implement it. To this end, we focus on alternative formalisms for specifying applications and platforms as well as different views on how a mapping can be described. In addition, we elaborate on the possible analysis and design-space exploration capabilities that various Y-chart based design methodologies support and how they assist a designer in finding a suitable solution for a design problem. We have chosen to write this paper because there are few papers which compare various design approaches for the implementation of the Y-Chart paradigm.

The objective of this paper is not to offer the reader an in-deep comparison of various implementations in order to determine ranking. The objective is rather to present and discuss some interesting design approaches the authors have observed for implementing the different steps of the Y-Chart paradigm. These approaches will be compared in order to highlight capabilities, limitations and trade-offs. The authors would also like to stress that great emphasis will be put on design and not implementation of these approaches, hence issues such as performance and scalability will not be greatly discussed. The three specific implementations of the Y-Chart paradigm used in this paper are Metropolis, the Distributed Operation Layer incorporating Modular Performance Analysis and the Y-chart variant of the Software/Hardware Engineering methodology. The paper also presents the history and key underpinning concepts of the Y-Chart paradigm in order to offer an in-deep presentation of the paradigm. The secondary objectives of this paper are to offer a good introduction to the Y-Chart paradigm as well as contribute to the design debates surrounding its implementation.

This article is organized as follows. After presenting the background of the Y-chart paradigm in more detail, we summarize related work on various design methodologies that implement it. Subsequently, we elaborate on formalisms for describing applications and for specifying hardware platforms. After a detailed overview of three particular methodologies, we give a concise comparison that highlights alternative approaches for implementing the Y-chart paradigm. The examination and comparison of the methodologies show that the implementation of the Y-chart paradigm maybe done in various ways.

2. BACKGROUND

2.1 Y-Chart Based Design-Space Exploration

The Y-chart paradigm is an example of a model-driven engineering framework [Bezivin 2005]. The crux of explicitly separating applications (functionality) from the platform (architecture) as shown in Figure 1 leads to the following five-step approach to minimize overall design time:

(1) Create a model of the functionality performed by each application in a fashion that is independent of any specific platform and which expresses opportunities for parallel execution;
(2) Create a model of the platform that captures key characteristics of the services it can provide to applications using resources like processors, busses, memories and power; as well as the cost of these services in terms of area, performance, energy, etc;

(3) Define a mapping of how the platform is deployed to execute the applications;
(4) Evaluate the quality of the parallelized applications mapped onto the platform in terms of area, performance, energy, etc and decide on improvements for the application software, platform design or mapping in case the results are unsatisfactory. If so, repeat step 1, 2 or 3 respectively according to the new insights until a solution is found that satisfies all design requirements;
(5) Realize the solution in terms of synthesising hardware and compiling software.

Y-chart based design methodologies assist designers by providing : (i) a coherent set of formalisms, (ii) techniques for analyzing design solutions and (iii) techniques for searching the design space. They often offer computer aided design tools. Formalisms refer to languages for writing down models. To efficiently exploit platforms with multiple processing elements, it is necessary to make potential parallelism in applications explicit. *Models of Computation* (MoC) are formalisms that describe systems in terms of computation and communication entities. Computation entities (or tasks) capture functional behaviour, whereas communication entities represent data and control dependencies between concurrently executed tasks. Most MoCs will help designers to make explicit potential parallelism. An application model created in step 1 expresses the services required for executing functional behaviour. These service requests are often characterized by the number of instructions to execute or the amount and size of data to be communicated. The term *Model of Architecture* (MoA) is sometimes used to denote a MoC that is specifically intended for describing platforms. A platform model for step 2 specifies the services that the resources of a platform can provide as well as the cost of using them (cycles, energie, etc.). Notice that although we use MoC and MoA to emphasize the purpose of formalisms, Y-chart based design methodologies may actually use the same formalism for describing both applications and platforms. A mapping in step 3 basically represents a possibly interesting match between the service requesting computation and communication entities in applications and the service providing resources in a platform. When a specific combination of application models, a platform model and a mapping yields satisfactory results with respect to the design requirements, then step 5 realises this particular design solution using appropriate software and hardware synthesis tools.

The Y-chart paradigm is very suitable for design problems where platforms offer multiple resources in order to execute a particular computation. The suitability is a consequence of the ease of defining various mappings between an application and a platform which are defined separately. For design problems where mapping alternatives are fairly absent, such as in telecommunication networks or traditional control systems, applying the Y-chart paradigm can be counterproductive. The reason for this is that such systems actually benefit from a higher degree of coupling between the functionalities they offer and implementation of latter. Methodologies supporting the development of such integrated models include SystemC based approaches [Grotker et al. 2002], System-on-Chip Environment [Gajski et al. 2000] and Software/Hardware Engineering [Theelen et al. 2007]. These methodologies often rely on successive refinement of the integrated model towards a synthesisable design where a distinction between applications and the platform on which they run is less prominent than in the Y-chart paradigm.

2.2 Models of Computation

An interesting view on MoCs is given in [Burch et al. 2001]:

"A model of computation is a distinctive paradigm for computation, communication, etc. For example, the Mealy machine model of computation is a paradigm where data is communicated via signals and all agents operate in lockstep (we use "agent" as a generic term that includes both hardware circuits and software processes). The Kahn Process Network model is a paradigm where data carrying tokens provide communication and agents operate asynchronously with each other (but coordinate their computation by passing and receiving tokens). Different paradigms can give quite different views on the nature of computation and communication. In a large system, different subsystems can often be more naturally designed and understood using different models of computation."

Having different views on the nature of computation and communication implies that modeling an application with various MoCs may result in models that differ significantly in the amount of details that is expressed about computation and computation. In the context of the Y-chart paradigm, MoCs that explicitly express (potential) concurrency between computations are of special interest. We categorize MoCs as control-oriented, dataflow-oriented and process-oriented:

—*Control-oriented* MoCs consider systems as automata, which consist of states and transitions between these states. A state captures a certain status reachable by executing a (collection of) computation(s), while transitions describe possible changes in this status (i.e., the execution steps). Although automata are mostly suited for describing pure sequential systems, some control-oriented MoCs such as Communicating Finite State Machines (CFSM) [Brand and Zafiropulo 1983] and Co-design Finite State Machine (CDFSM) [Balarin 1997] allow describing a system as a collection of concurrent state machines that can communicate with each other.

—*Dataflow-oriented* MoCs describe systems in terms of tasks (actors or processes), channels and tokens. A task is a computation entity that can potentially be executed in parallel with other tasks. Tasks communicate with each other by exchanging tokens through channels. Such a token denotes an indivisible unit of information. The channels often include FIFO buffers to enable the sending and receiving tasks to run at a different rate, while successively communicated tokens are processed in-order.

—*Process-oriented* MoCs use processes and events as major modeling entities. A process represents a computation entity that may be executed in parallel with other processes. Processes can synchronise based on communication events, which can for example be in terms of signals or passing messages.

It is sometimes possible to express a MoC of one type by using a MoC of another, however such alternative representations are not always very intuitive. Such an example is the process-oriented MoC of SystemC [Grotker et al. 2002] that can express various control-oriented, dataflow-oriented and other process-oriented MoCs. Another such example is use of certain types of automata (i.e., control-oriented MoCs) to formally express the semantics of dataflow- and process-oriented MoCs such as Synchronous Dataflow [Lee and Messerschmitt 1987; Ghamarian 2008] and the Par-

	MoC	Reference	Concurrency	Communication	Time	Explicit Non-Determinism	Stochasticity	Analytic Tractability
Control-Oriented	Finite State Machines	[Hopcroft and Ullman 1979]	-	-	Integer-valued*	-	-	++
	Discrete-Time Markov Chains	[Chung 1967]	-	-	Discrete real-valued distributions	-	Discrete distributions on time	++
	Timed Probabilistic Systems	[Alur 1991]	-	-	Discrete real-valued distributions	+	Discrete distributions on behavior and time	+
Dataflow-Oriented	Synchronous Dataflow	[Lee and Messerschmitt 1987]	Asynchronous	Asynchronous FIFO buffered, token-driven	Integer-valued*, discrete integer-valued distributions*	-	Discrete distributions on time*	++
	Cyclo-Static Dataflow	[Bilsen et al. 1995]	Asynchronous	Asynchronous FIFO buffered, token-driven	Integer-valued*	-	-	++
	Scenario-Aware Dataflow	[Theelen et al. 2006]	Asynchronous	Asynchronous FIFO buffered, token-driven	Discrete real-valued distributions	+*	Discrete distributions on behavior and time	++
	Boolean Dataflow	[Buck 1993]	Asynchronous	Asynchronous FIFO buffered, token-driven	-	-	-	o
	Kahn Process Networks	[Kahn 1974]	Asynchronous	Asynchronous FIFO buffered, token-driven	-	-	-	--
	Dynamic Dataflow	[Buck 1993]	Asynchronous	Asynchronous FIFO buffered, token-driven	-	+	-	--
	Reactive Process Networks	[Geilen and Basten 2004]	Asynchronous	Asynchronous FIFO buffered, token-driven and synchronous control events	-	+	-	--
	Real-Time Calculus	[Thiele et al. 2000]	Asynchronous	Asynchronous (FIFO) buffered, data-driven	Continuous	+	+*	++
Process-Oriented	Esterel	[Boussinot and Simone 1991]	Synchronous	Unbuffered asynchronous signals	Discrete integer valued	-	-	+
	Communicating Sequential Processes	[Hoare 1978]	Asynchronous	Synchronous message passing	-	-	-	++
	Communicating Concurrent Systems	[Milner 1989]	Asynchronous	Synchronous message passing	-	-	-	++
	Metropolis Meta Model	[Balarin et al. 2003]	Asynchronous	Unbuffered asynchronous events	Discrete real-valued	+	-	o
	Timed Automata	[Alur and Dill 1994]	Asynchronous	Synchronous message passing	Discrete real-valued or continuous	+	Discrete distributions on behavior and discrete or continuous distributions time*	+
	Parallel Object-Oriented Specification Language	[Bokhoven 2002]	Asynchronous (two levels)	Synchronous message passing	Discrete real-valued	+	Discrete distributions on behavior and time	+

Table I. Comparing some example models of computation

allel Object-Oriented Specification Language [Bokhoven 2002]. Some research has been done on unifying various MoCs. Such unification is important when defining subsystems with different MoCs. Examples of unifying MoCs are the Tagged Signal Model Framework (TSM) [Lee and Sangiovanni-Vincentelli 1998] and the Ptolemy Project [Brooks et al. 2005].

Table I gives an overview of a number of widely used MoCs. It summarize their approach to concurrency, communication and time. These aspects are essential for modeling distributed embedded system. A * indicates that the aspect is supported by some variant of the MoC. The table also indicates whether the abstraction mechanisms of explicit non-determinism (i.e., not implied by concurrency but by some other language construct) and stochasticity are supported. The last column gives some impression to what extend the basic form of a MoC is analytically tractable. Comparing MoCs on this last aspect is very difficult and the results in Table I should therefore be taken with precaution. A -- indicates that (nearly) all properties are undecidable at design time. In case some design-time analysis is possible, such as structural consistency checks, a - is used. Conversely, ++

indicates that (nearly) all models expressed in the considered MoC are fully design-time analyzable, both for correctness and performance. A + expresses that not all but many models are fully analyzable. A problem could for example be the implication of an infinite state space by certain models. A o denotes analytical tractability between - and +.

2.3 Models of Architecture

As the previous section suggests, the concept of MoCs is very mature. Conversely, the concept of MoAs has only started to emerge in the last decade. Although there exists no formal definition of the term, two major views on MoAs can be identified:

(1) Describe platform components using existing MoCs;

(2) Specify platforms using dedicated formalisms.

The earliest uses of the first approach are Cadence Alta VCC [Martin 1998] and POLIS. Others examples are SystemC-based solutions, Metropolis [Balarin et al. 2001] and the automated design flow of [Stuijk 2007]. Each of these uses the same modeling constructs to define applications and platform models.

Dedicated formalisms for specifying platforms are also known as Architecture Description Languages (ADL) [Medvidovic and Taylor 2000; Mishra and Dutt 2008]. It is very difficult to categorize ADLs (or MoAs) in a manner similar to Table I since very few ADLs focus on the same platform types or on the same platform details. [Qin and Malik 2002] presents major contributions for ADLs to describe general-purpose processors. [Gries and Keutzer 2005] presents Mescal, an ADL focused on Application-Specific Instruction-set Processors (ASIP). The IP-XACT Standard (www.spiritconsortium.org) [Kruijtzer et al. 2008] is an example of an ADL that focuses on IP-based platforms. Other examples are proprietary languages that configure certain template descriptions in another MoA or MoC. These typically require special compilers to expand (generate) a full specification. Examples of this approach are Colif [Cesario et al. 2001] and the XML specifications for the tool in [Theelen 2007].

In the context of distributed embedded systems, many researchers recognized processing (e.g., general-purpose processors, accelerators and dedicated controllers), communication (e.g., busses, network-on-chip, and i/o interfaces) and storage (e.g., memories and hard disks) resources as elements of a MoA. The need for storage resources originates from using a certain processing or communication resource, which means that storage resources only provide a service to applications via these resources. Another indirectly provided resource is power (or energy), which is consumed by any of the other resource types. Figure 2 illustrates the hierarchical relation between the services that resources provide to each other and to the service requesting computations and communications of applications through mappings. Notice that sharing a processing, communication and storage resource requires scheduling the moments at which each of the involved service requesting entities accesses this resource. Depending on the exact evaluation criteria a designer is interested in (area, performance, energy, etc), a MoA should allow describing the indicated four resource types, thereby taking the service providing relations between resource types and potential contention due to resource sharing into account. Notice that the service providing relations between the different resources are in

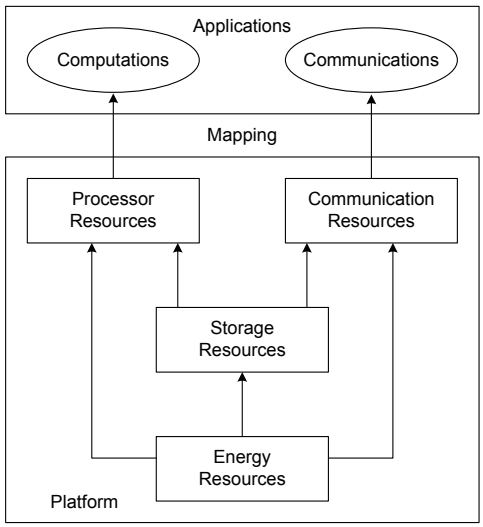


Fig. 2. Hierarchy of providing services between resources in platforms to applications

reality realized by physical connections (in hardware). A MoA may or may not require specifying the physical architecture of how the resources are interconnected, see also Section 4.2. The next section discusses some examples of special languages for describing the services provided by platforms.

3. METHODOLOGIES AND LANGUAGES

The Y-chart paradigm has been integrated into various design methodologies. The earliest well known implementations are Cadence VCC and POLIS. Other well-known examples include SPADE [Lieverse et al. 2001], Daedalus[Thompson et al. 2007] and SymTA/s [Hamann et al. 2004]. This section presents three implementations that will be used for design comparison. This particular selection was made because each implementation proposes some interesting design approaches. Metropolis differentiates itself from the others by its strict adaption of the Y-Chart paradigm. Moreover, it uses an interesting combination of informal specification for models and formal specifications for mappings. The Y-chart variant of the Software/Hardware Engineering methodology differentiates itself by adapting an existing methodology in order to incorporate the Y-Chart paradigm. Moreover, it uses a single formal language for both modeling and mappings. The Distributed Operation Layer incorporating Modular Performance Analysis differentiates itself by using Modular Performance Analysis which permits high-level performance analysis.

3.1 Metropolis

The Metropolis project [Balarin et al. 2003] has been running since 1999 and is a joint initiative of the Gigascale Silicon Research Center, the University of California and the Cadence Berkeley Laboratories. The project focuses on the modeling and the design of systems using a platform-based approach, as well as on the integration

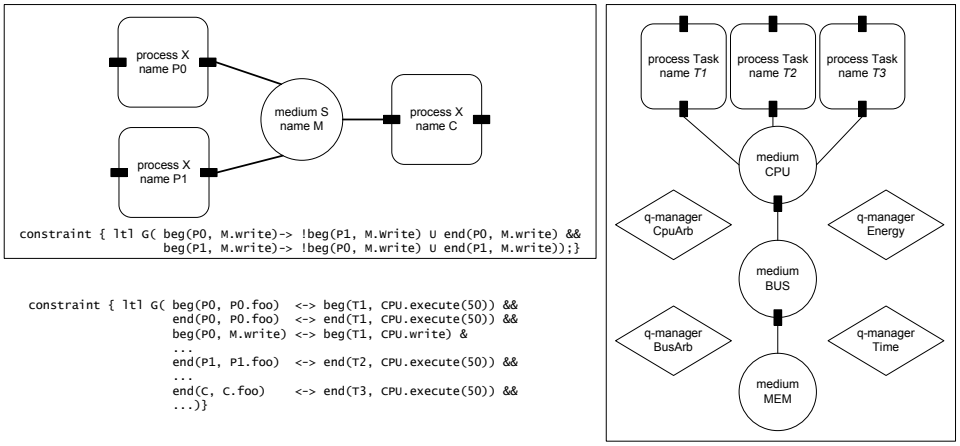


Fig. 3. Metropolis example (application model, platform model and mapping)

of multiple MoC in order to describe heterogeneous systems at various abstraction levels [Sgroi et al. 2001].

The primary objective of Metropolis is to enable the development of design flows for different application domains. To achieve this objective, Metropolis promotes separation of concerns and platform-based design [Pinto 2004]. It strongly relies on the Y-chart paradigm and therefore follows its five step design approach explicitly. Metropolis advocates the concept of reuse by explicitly decoupling the specification of independent aspects over a set of abstraction levels. Other research has focused on a compositional modeling methodology around Metropolis [Goessler and Sangiovanni-Vincentelli 2002]. The Metropolis framework [Davare et al. 2007] consists of three main elements:

—An infrastructure which encompasses the Metropolis Meta-Model formalism, a compiler and an Abstract Syntax Tree specification;

—A Meta-Model Library for MoCs and architecture platforms;

—A tool set for simulation, verification and synthesis.

The Metropolis-Meta-Model (MMM) forms the core of Metropolis. It is a Java inspired language that adds the necessary semantic and syntactical elements for system design (i.e., processes, communication channels, ports, etc.). MMM supports some novel features such as denotational formulas in Linear-Temporal Logic (LTL) and the predicate Logic of Constraints (LOC)[Balarin et al. 2001; Burch et al. 2001]. Moreover, the event model of MMM is based on the Tagged Signal Model Framework (TSM) [Lee and Sangiovanni-Vincentelli 1998]. Metropolis is not a formal method in the sense that it is not based on strict mathematical definitions. However, portions of the MMM language such as the LTL and LOC expressions are formal. These can be verified with model checking technologies [Yang et al. 2006]. Having said this, the MMM language does have precise semantics for modeling and execution. To integrate formal methods into Metropolis, mechanisms are incorporated in MMM that enable tools to process suitable subsets of a design. The work in [Densmore 2004a] is an example of this principle for control graph analysis.

Metropolis has been applied in many academic and industrial case studies ranging over various domains such as automotive, wireless multimedia, analog/mixed signal systems and micro-processor design [Sgroi et al. 2001; Meyerowitz 2004; Zeng et al. 2006]. Similar to SystemC-based methodologies, Metropolis scales to fairly complex systems.

3.1.1 *Application Modeling.* Conforming to the Y-chart paradigm, Metropolis promotes developing application models which are called functional descriptions. As mentioned previously, Metropolis encourages a platform-based design flow, hence an application model is considered as a platform (functional platform) with a high degree of abstraction [Pinto 2004]. In the Metropolis literature, the term *denotational definition* is often used for functional descriptions. The MMM language is based on the TSM denotational framework which has been proven to support a vast amount of MoCs [Lee and Sangiovanni-Vincentelli 1998]. In [Densmore et al. 2006], it is suggested that Metropolis can support any (non-stochastic) MoC.

The basic concepts of MMM [Balarin et al. 2003] are processes, interfaces, ports, events and media. A *process* represents a sequential program and is also called a thread. A process communicates with its environment through one or more *ports*. A port is specified with an *interface*, which refers to a communication contract for exchanging information with the environment. Interfaces are implemented with *media*. Once a network of connected processes and media is defined, like the example shown in Figure 3, the behavior of the network can be specified with the concept of *events*, which represent specific behavioral actions in the application model.

Once an application model is completed, it is possible to add various constraint specifications by using LTL and LOC expressions to it as illustrated in Figure 3. LTL formulas specify coordination constraints between processes. LOC formulas describe performance constraints. The Metropolis framework provides a library of functional platforms elements in YAPI [Kock et al. 2000] and TTL [Pinto 2004]. Metropolis also offers some support for using non-determinism as an abstraction mechanism. Two constructs exist; non-determinate variable assignment and a limited form of non-deterministic code execution.

3.1.2 *Platform Modeling.* As indicated, Metropolis describes application and platform models both as platform models in MMM but at different levels of abstraction. Where application models are concerned with the definition of functional aspects, they do not define any resource utilization aspects. Resource utilization aspects are expressed in platform models. Nevertheless, platform models are defined with the same primitives as application models (processes, interfaces, ports, media and events) capturing the functionality they offer and the cost (efficiency) of that functionality. Platform functionalities are modeled as services (methods) defined in interfaces. The cost of a service is modeled by associating events with the various portions of the implementation of a service and then annotating each event with a value that represents its cost. With the language primitives of MMM, any resource type can in theory be modeled. References [Balarin et al. 2003; Pinto 2004; Meyerowitz 2004] illustrate the modeling of processor, communication and storage resources. The way in which resources are interconnected must be modeled explicitly. To take aspects such as scheduling and power consumption into account,

the concept of quantity managers is introduced. Quantity managers are responsible for the assignment of tags to events and for the ordering of events. Figure 3 shows an example of a platform composed of a single processor, a bus and a memory. The computation resources of the processor are modeled explicitly using tasks.

An important aspect of the platform-based design approach followed by Metropolis is the ability to refine high-level platform models into more detailed ones. The MMM language supports this by providing primitives for declaratively specifying that a particular model is a refinement of an element or a group of elements of another model. The primitives also enable to specify in detail how the original model can be replaced by the more detailed model to achieve refinement [Balarin et al. 2003; Densmore et al. 2004; Densmore 2004b; 2004a]. The latter reference discusses a formal approach to verify that a refinement preserves certain properties of the original model.

3.1.3 *Mapping.* Metropolis offers a novel way of mapping an application model to a platform model based on formal synchronization expressions. LTL is used to specify these expressions. Platform models may contain non-determined values, such as the memory address of a variable or the priority of a task. These non-determined values are fixed by means of value mappings in the synchronization expressions. Hence, it is possible to map a value from an application model to a value in the platform model during synchronization. Figure 3 shows an example of mapping expressions.

3.1.4 *Evaluation (Analysis and Exploration).* The primary vehicle for analysis in Metropolis is simulation. Two simulation tools are available for MMM models, which respectively generate SystemC or pure C++ specifications for simulation purposes. The compiler provided with Metropolis is also capable of generating monitors in order to verify LTL and LOC expressions during simulation [Balarin et al. 2003; Yang et al. 2006]. The work in [Chen et al. 2005] discusses a simulation-based deadlock analysis technique. Any quantities that have been explicitly added to a model can be traced and analyzed after a simulation. Instead of using simulation, parts of an MMM model can be formally verified using model checkers. Metropolis offers a tool to produce PROMELA code from an LTL expression, which can then be verified using SPIN [Yang et al. 2006]. The framework also provides a LOC expression checker [Balarin et al. 2003].

The design-space exploration step of the Y-chart paradigm is not automated in Metropolis [Zeng et al. 2006]. Designers must manually create alternative platform models and mappings for an application model and use the supported analysis techniques for evaluation. The work in [Balarin et al. 2003] presents a *quasi-static* scheduling technique in order to schedule a concurrent specification on shared resources. One may consider that the technique allows automatic design-space exploration for shared resource utilisation.

3.2 Distributed Operation Layer and Modular Performance Analysis

The design framework DOL (Distributed Operation Layer) as described in [Thiele et al. 2007] closely follows the Y-Chart paradigm of Figure 1. It is targeted towards the mapping of applications to multi-processors on a chip (MPSoC). The approach is based on a very early implementation of the Y-chart paradigm pre-

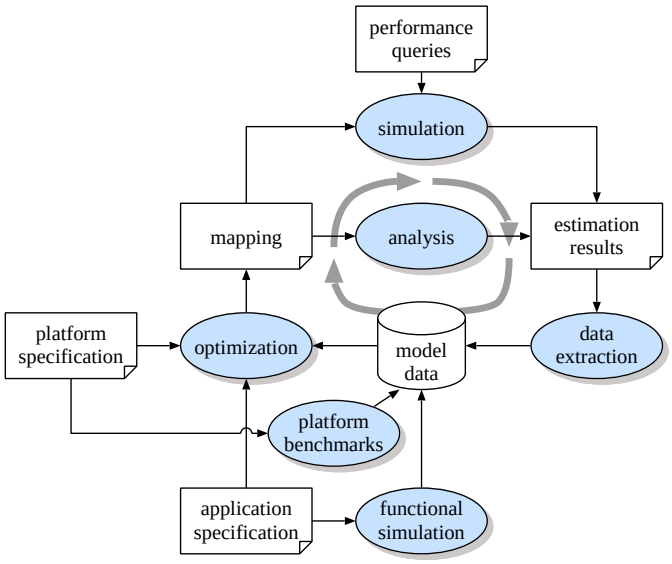


Fig. 4. Y-chart based optimization cycle in the DOL environment

sented in [Teich et al. 1998]. This latter approach uses graph structures to formally specify the application and target platform. The mapping is also modeled as a graph, which captures not only the binding information, but also scheduling. Only the feedback-edges regarding the improvement of the platform and mapping shown in Figure 1 are supported by [Teich et al. 1998], where the necessary optimizations are performed automatically based on multi-objective evolutionary algorithms. The DOL environment, see Figure 4, applies the same principle but with a much more refined application/platform modeling and analysis approach. In particular, the application is specified with a set of communicating tasks, where the communication structure is provided as an XML specification. The individual tasks are sequential programs that are given in a programming language for which an appropriate compiler is available; the API of DOL only provides the necessary semantical interfaces to the communication channels. In a similar way, the underlying hardware platform and its capabilities are described by an annotated graph structure, including processing and memory resources and their interconnection through communication resources like buses and networks. The mapping, again specified by means of an XML specification, relates tasks to computing resources and memories. It also links tasks to paths on the communication platform and specifies the applied resource sharing mechanism.

The analysis is performed using a hierarchy of various methods. The fastest evaluation of a system configuration is done using the Modular Performance Analysis (MPA) framework [Chakraborty et al. 2003; Wandeler et al. 2006], a compositional performance analysis method for heterogeneous distributed embedded systems. In addition, the necessary parameters are determined using a simulation-based profiling of the application and the underlying hardware platform. The multi-objective optimization is based on evolutionary algorithms and uses the PISA environment

[Bleuler et al. 2003] that is publicly available. The mapping information is used to generate highly efficient code for the target platform using a dedicated generation of the hardware-dependent software including calls to the underlying operating system. Finally, a trace-based simulation can be used to determine the extra-functional properties of mapped applications with high accuracy.

We now give a short overview of the theoretical core underlying the design-space exploration framework of MPA. It is a very efficient method due to the high-level of abstraction of the models that it uses. MPA is an analytical approach based on the Real-Time Calculus [Thiele et al. 2000], which has its foundations in methods for worst-case analysis of communication networks (Network Calculus) [Cruz 1991]. MPA is an example of exact performance analysis approaches that can determine guaranteed performance limits. While these techniques can compute hard performance bounds, they abstract from the complex interactions and state dependent behavior in the system. MPA uses a unifying model for the representation of different event patterns in the form of *arrival curves* known from the communication domain [Cruz 1991]. In addition, it uses a similar concept called *service curves* to represent the resources and their computational or communication capabilities, which allows MPA to model complex hierarchical scheduling schemes in distributed embedded systems. The detailed modeling of the capabilities of the shared resources and the event streams can lead to highly accurate performance results, see for example [Chakraborty et al. 2003].

An MPA model is a performance network of components, where application tasks are mapped to computation and communication resources. One can differentiate between three main entities: *event streams* represented as arrival curves, *resource streams* represented as service curves, and *application tasks* represented as processing components. Application tasks are activated by event streams which they process by considering the interaction of the event streams with the resource streams. On a higher level, the model is a network of components that communicate with each other through *event interfaces*. Performance metrics for the whole application are computed by combining the behavior of the individual components. This modularity aspect achieves short analysis times even for large systems. Typical performance metrics computed with MPA are upper and lower bounds on buffer levels, end-to-end delays experienced by events, and the number of events that can be processed in a time unit (throughput). MPA supports refinement of the elements of a performance network with the extension Real-Time Interfaces [Thiele et al. 2006]. It promotes interface-based design of embedded systems with the concept of adaptive interfaces.

The method has been implemented as a Matlab toolbox [Wandeler and Thiele 2006b] where a system is described as a Matlab script file. The toolbox implements the Min-Plus and Max-Plus algebraic operators and provides the facilities to represent the arrival and service curves, and the processing components. It also contains a library of predefined components that assist the designer in building a system performance model. Different case studies have been performed, covering for example car infotainment systems, MPSoC platforms for multimedia applications, digital signal processing systems and network processors.

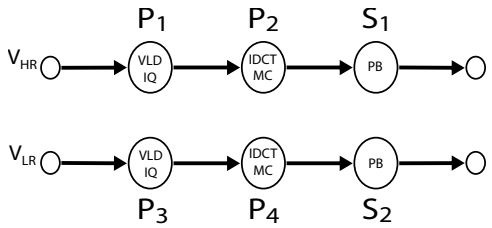


Fig. 5. Task graph description of an application processing two MPEG2 video streams in parallel

3.2.1 *Application Modeling.* Modeling an application in DOL-MPA involves capturing the application tasks and the event flows between them. The application task model provides information about the processing semantics of the tasks. An example of an application task graph for parallel decoding of two MPEG2 video streams is shown in Figure 5.

The goal of MPA is to analyse the timing behavior of an application considering a large class of possible event flow characteristics. Traditionally, the timing behavior of event flows is modeled as being periodic or periodic with jitter. However, such abstract representations cannot adequately capture the complex timing behavior of event flows in a highly parallel or distributed system. Hence, a more powerful abstraction is needed. Variability Characterization Curves (VaCC) substantially generalise the traditional representations and can express any possible timing behavior of an event stream. Event streams in MPA are captured by a special kind of VaCCs, denoted as arrival curves. They provide upper and lower bounds on the number of events in any time interval. For an event stream α, there are at most $\alpha^u(\Delta)$ and at least $\alpha^l(\Delta)$ events within any time interval $[t, t+\Delta)$ for all moments t. Figure 6 illustrates how arrival curves bound the behavior of a periodic event stream. Information about the arrival curves representing the interactions with the environment can come from several sources. Firstly, they can be computed analytically if an interaction has some pattern such as periodic, periodic with jitter, sporadic, etc. In case that they are unknown, they can be computed as an envelope of a set of recorded traces. Finally, they can be derived from specifications like UML sequence diagrams that describe the behavior of the event-generating devices.

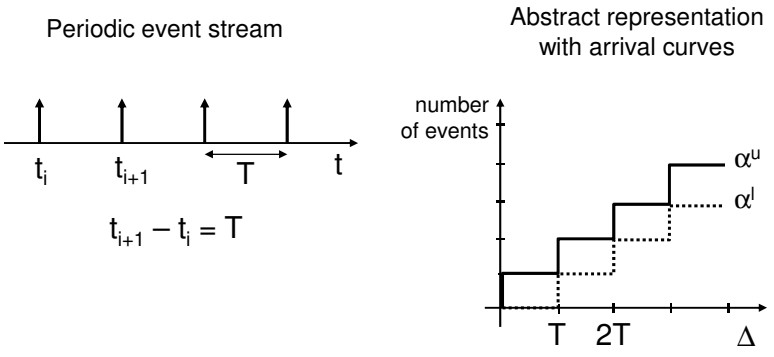


Fig. 6. Modeling periodic event streams in MPA

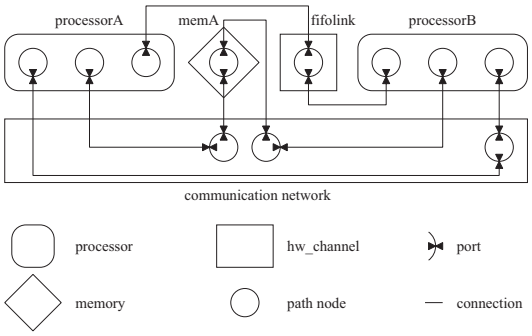


Fig. 7. Platform model in the DOL environment

For an accurate performance analysis, it is vital that the resource demands associated to individual events are modeled precisely. To this end, MPA supports the conversion between event-based arrival curves and resource-based arrival curves. Besides simple models based on best-case and worst-case behavior, automata can be used to model arrival patterns of different event types and the corresponding resource requirements [Wandeler and Thiele 2007]. A similar approach is used in MPA to model state-dependent workload demands as introduced by cache memories [Chakraborty et al. 2007].

3.2.2 *Platform Modeling.* The DOL environment extracts non-functional properties of the platform and builds abstract models of the resource services offered by the platform based on the MPA model. More specifically, MPA models the resource capabilities of all computation and communication resources and it can provide information on how these capabilities are affected by the workload of tasks and communications. Resources in MPA are modeled explicitly and therefore considered 'first class citizens' of the approach. Figure 7 shows a simple platform specification in the DOL environment.

Resource capabilities, like event streams, can be described with VCCs. The service curves $\beta^u(\Delta)$ and $\beta^l(\Delta)$ provide upper and lower bounds on a service β within any time interval $[t, t+\Delta)$ for all moments t. The unit of service depends on the kind of resource, for example instructions or cycles for computation, and bytes for communication. The service curves of a resource can be determined using data sheets, analytically derived properties, or by measurements. Figure 8 illustrates the service curves that bound the service offered to a single task by a single slot in a Time Division Multiple Access (TDMA) resource. Using service curves, MPA can model any arbitrarily complex resource capabilities and is able to model arbitrary scheduling hierarchies.

3.2.3 *Mapping.* In a real-time system, an incoming event stream is usually processed by a set of Hardware/Software components. After the mapping of tasks to computing resources and streams to communication paths in the DOL environment, a performance model of the system is determined. Based on the MPA method, this performance model is a network of performance components where each of them has as inputs abstract event streams and abstract resource streams. More specifically, a

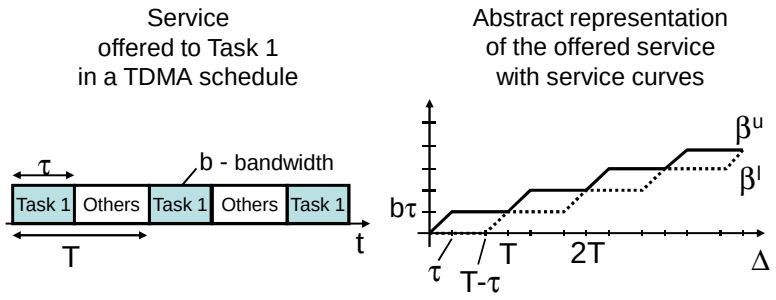


Fig. 8. Modeling a TDMA service in MPA

performance component defines equations for functional transformations of arrival and service curves where the actual equations depend on the processing semantics of the modeled task. In other words, the abstract resource streams interact with the abstract event streams in a performance component. This performance component produces as outputs a transformed abstract event stream and a remaining abstract resource stream that is available to other tasks mapped onto the same resource. The mapping and the respective MPA performance network of the task graph from Figure 5 are shown in Figure 9.

Given a specific mapping, MPA also needs information about the workloads induced by the mapped application tasks running on the specific resource. This information is needed by the performance components for the transformations of arrival and service curves. Usually, it represents upper and lower bounds on the service needed by the component to process one, two, and more consecutive events from the incoming event stream. Such information can come from cycle-accurate simulations of the application tasks or from static analysis of the program code and the chosen hardware architecture [Wilhelm et al. 2008], see also Figure 4.

If several tasks are mapped to the same resource, a resource sharing policy needs to be determined. Scheduling in MPA is modeled by the way performance components are interconnected. Supported scheduling policies are preemptive and non-preemptive fixed priority, TDMA, earliest deadline first (EDF), generalised processor sharing (GPS), first-in first-out (FIFO), hierarchical scheduling, and different server strategies [Wandeler and Thiele 2006a]. An example for modeling of preemptive fixed priority scheduling are tasks $P2$ and $P4$ in Figure 9 sharing $CPU2$.

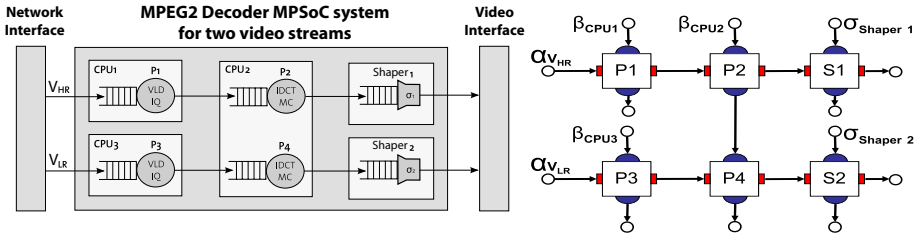


Fig. 9. Specifying a mapping configuration as an MPA performance network

3.2.4 *Evaluation (Analysis and Exploration).* The DOL design environment closely follows the Y-chart approach. The exploration is done using a multi-objective evolutionary optimization approach using the PISA platform [Bleuler et al. 2003]. In terms of performance estimation, DOL is not bound to a particular method. It can use analytical methods such as MPA or simulation-based ones such as functional and trace-based, see Figure 4.

MPA can determine the characterisations of all event and resource streams in the network of performance components using the abstract characterisations of the workloads and the input event and resource streams. From the computed arrival and service curves, MPA deducts information about the utilisations of the resources, the end-to-end delays between any two components, the necessary buffer spaces for event and packet queues, and the throughput. The modularity, efficiency and scalability of the MPA models makes the method highly suitable for quickly analysing a large number of different mappings and resource sharing policies during design space exploration. Extension of applications by adding tasks is modeled by simply adding components in the performance network. A single performance model can include different resource sharing strategies without affecting the accuracy of the performance analysis results.

MPA provides hard upper and lower bounds on the computed performance results. However, it is a worst-case approach that covers all possible corner cases regardless of their probability of occurring. Even if the results can be very close to simulation results [Chakraborty et al. 2003], in some cases a designer is interested in the average case behavior of a distributed embedded system. In this sense, MPA is a complementary method to other simulation-based or stochastic analysis techniques. It is able to analyse the timing non-determinism of complex distributed embedded systems while providing hard guarantees on the resulting end-to-end behavior.

3.3 Software/Hardware Engineering

Originally introduced in [Putten and Voeten 1997], Software/Hardware Engineering (SHE) evolved into an industrial-strength system-level design methodology accompanying many methods, techniques and tools for the design, analysis and synthesis of distributed real-time hardware/software systems [Theelen et al. 2007]. SHE considers the Y-chart paradigm as a way to specialise its generic model-driven engineering framework to facilitate a flexible approach towards multi-processor design for streaming multi-media applications [Wijk et al. 2003; Florescu et al. 2007; Theelen 2008]. Because the provided methods, techniques and tools are not specifically targeted to the Y-chart paradigm, SHE is more generally applicable than methodologies that enforce a separation between application and platform models.

SHE is built around the Parallel Object-Oriented Specification Language (POOSL), which is a very expressive general-purpose modeling language with a process-algebraic formal semantics [Bokhoven 2002]. It includes powerful primitives for intuitively describing (hierarchical) structure, concurrency, communication, data, time and stochasticity. POOSL distinguishes three types of object classes originating from the idea of modeling active and passive system components separately. *Data objects* represent passive information that is generated, communicated, processed, etc. by active components, which are modeled with processes and clusters. *Processes* represent basic active components that may initiate both se-

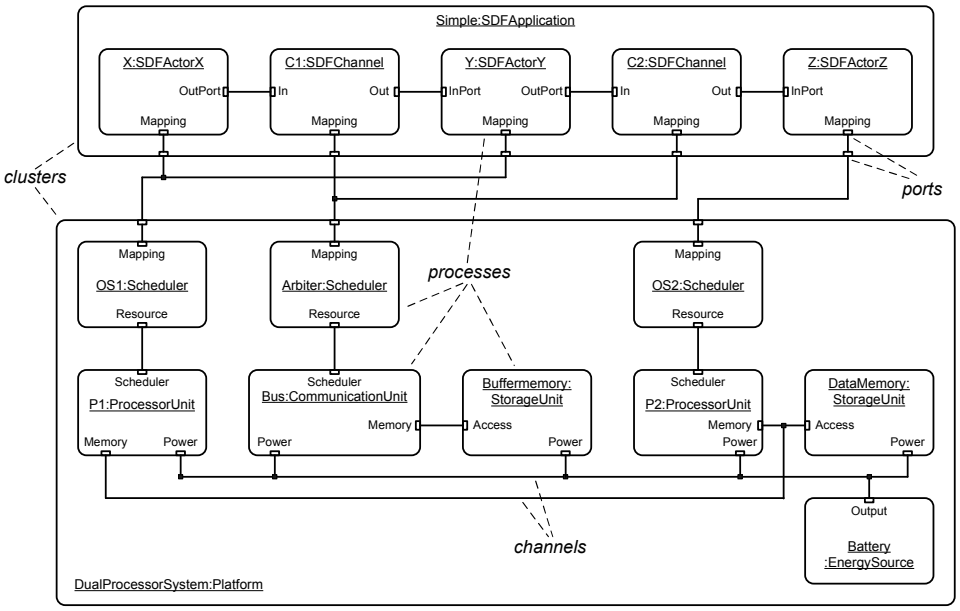


Fig. 10. Instances of process and cluster class modeling patterns capturing a streaming system

quential and concurrent behavior. *Clusters* allow describing hierarchical structures between active components (processes and clusters). Processes and clusters can communicate with each other by passing messages over channels through ports. Such messages may include data objects to specify the exchange of information. Figure 10 illustrates a structure of active components in POOSL, where a simple application with 3 tasks (actors) is mapped onto a battery-powered dual-processor platform.

The mathematically defined semantics of POOSL is the crux in supporting

—Interactive simulation of models [Geilen 2002; Bokhoven 2002] to facilitate validation of whether a model adequately represents the system under design;

—Model checking and simulation-based analysis of functional correctness [Geilen 2002] and performance [Theelen 2004] like absence of deadlock and throughput;

—Generation of real-time control software [Huang 2005], which relies on a step-wise refinement approach that guarantees preservation of functionality and timing.

Combining the Y-chart paradigm with SHE currently exploits only the first two aspects and hence does not cover the step of synthesising the final design solution (see also Section 2.1). The Y-chart variant of SHE is primarily based on modeling patterns. *Modeling patterns* are parameterised model components for capturing typical aspects of systems in a certain modeling language [Theelen 2004]. In this case, it refers to a collection of template data, process and cluster classes targeted to the application domain of streaming multi-processor systems. Several modeling patterns have been developed to ease constructing performance models for Y-chart based design-space exploration. Hence, these modeling patterns cover both application modeling and platform modeling. A specialised tool called PREMADONA

automates constructing POOSL models by properly instantiating the modeling patterns from an MPSoC specification given in XML [Theelen 2008].

SHE has been applied in many academic and industrial case studies [Theelen et al. 2007] ranging from communication protocols, internet routers, television systems, car-infotainment systems, network-on-chip based multi-processors, printer systems up to wafer scanners. Although model checking has proven to be feasible for relatively small systems, most case studies relied on simulation-based analysis. The simulation tools of SHE have shown to be competitive with tools like OPNET and SystemC in terms of simulation speed. Analysing models with over 10^6 parallel processes demonstrated scalability to systems of industrial complexity.

3.3.1 *Application Modeling.* Modeling applications following the Y-chart variant of SHE is based on capturing service requesting behaviors, such as those shown in Figure 2, in POOSL. Because of its expressive power, any MoC can in essence be represented, even when including all functional details. However, considering the Y-chart's focus on performance analysis, expressing applications in POOSL urges to abstract from functional details, which conforms to SHE's strategy. Relevant aspects like the structure of how computations interact with each other and resource requirements like execution times and memory usage must be taken into account. To ensure an adequate representation of any dynamism in applications, one may even use probabilistic and non-deterministic approaches as abstraction mechanisms. Moreover, POOSL allows data or control events to originate from files, which facilitates evaluating how a system reacts for example to observed user interactions.

To ease using the Y-chart paradigm with SHE, several POOSL modeling patterns have been developed for MoCs that abstract from functional details but still allow annotations with key resource requirement characteristics similar as to what SHE advocates for performance modeling. These include the dataflow-oriented MoCs of Synchronous Dataflow (SDF), Cyclo-Static Data-flow (CSDF) and Scenario-Aware Dataflow (SADF) [Theelen 2008]. The top cluster in Figure 10 shows actually a combination of the modeling patterns for an SDF application. For time-driven and event-driven task specifications similar to those common in traditional scheduling theory, POOSL modeling patterns have been developed that probabilistically mimic the behavior of such tasks for uncertainties regarding activation latency, release jitter and output jitter [Florescu 2007]. Figure 11 illustrates how the modeling patterns for SDF capture the typical interaction between computation and communication entities in such applications. It also shows how the scheduler of a platform acknowledges a request for executing a computation.

Although modeling patterns form the crux to Y-chart based design with SHE, specifying POOSL may not be the most convenient way of constructing application models. Therefore, the more intuitive XML specification formats for specifying SDF, CSDF and SADF models defined in [Stuijk 2007; Theelen 2007] have been adopted as input language for the model generation tool PREMADONA. Similarly, task graphs consisting of time-driven and event-driven tasks can be described in the XML format defined in [Florescu 2007]. Given such an XML specification, PREMADONA automatically instantiates the appropriate combination of the patterns.

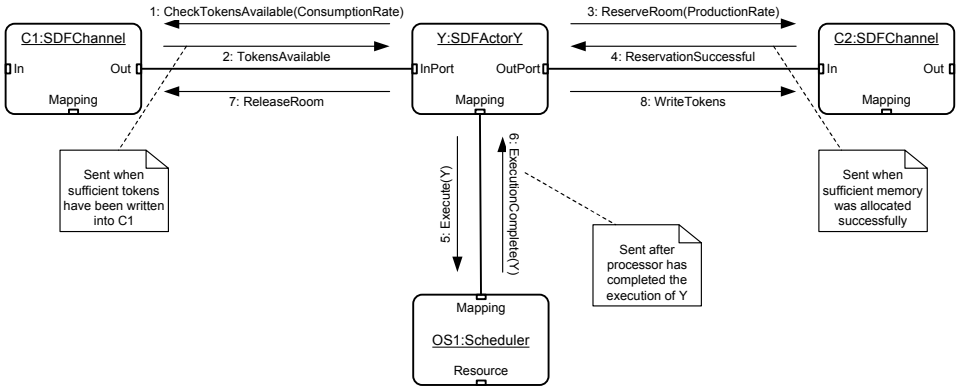
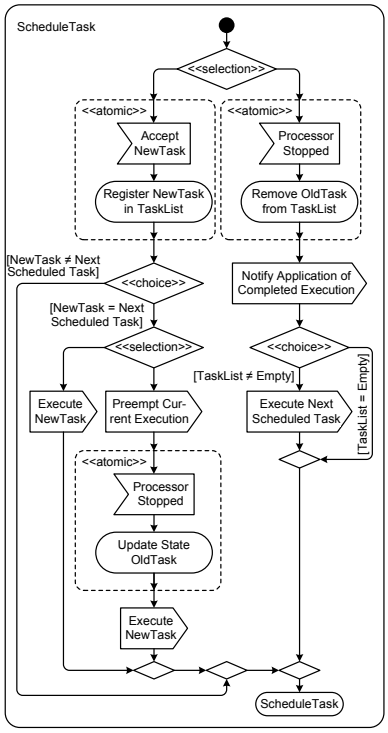


Fig. 11. Example of an interaction between an SDF application and a scheduler of a platform

3.3.2 *Platform Modeling.* Modeling platforms with SHE is based on capturing service providing behavior in POOSL. In essence, any resource type can be expressed, even at a fully synthesizable cycle-accurate register transfer level. However, since the Y-chart paradigm aims at performance evaluation of design alternatives, SHE advocates abstraction from implementation details; hence, only capturing crucial aspects that affect performance. These include the number and type of resources, the way in which they provide services to each other, and any aspect related to contention that arises from sharing resources, including any scheduling or arbitration mechanisms. To ensure an adequate representation of contention when abstracting implementation details, one may use probabilistic and non-deterministic approaches. These allow capturing technology-dependent uncertainties like unreliable communication media or deep submicron issues.

Various POOSL modeling patterns have been developed to represent all four resource types (processor, communication, storage and energy resources) of MPSoCs together with various types of non-preemptive and preemptive schedulers that can be used when sharing processor and communication resources [Theelen 2008; Florescu 2007]. In addition, a basic resource manager is available [Kumar et al. 2006]. Figures 12 and 13 show some behavioral details of two modeling patterns using activity diagrams and the corresponding POOSL code. The PREMADONA tool instantiates the modeling patterns when generating platform models from an XML specification that contains key parameters like clock frequencies, voltage/frequency scaling factors, and power consumption characteristics. The current collection of modeling patterns together with the XML specification format give a MoA for describing relatively simple network-on-chip based multi-processor platforms, which differs in various ways from other approaches such as those in [Balarin et al. 2003; Gries 2004; Thompson et al. 2007]. An example of such a difference is that no actual data processing is done, nor are instruction sets emulated (i.e., execution of the model does not provide a functional result). It is also not required to specify for example the size of memories or the number of concurrent connections that a network-on-chip can realize. These aspects are considered to be a result of the evaluation step. The abstraction goes even further in not requiring to specify how



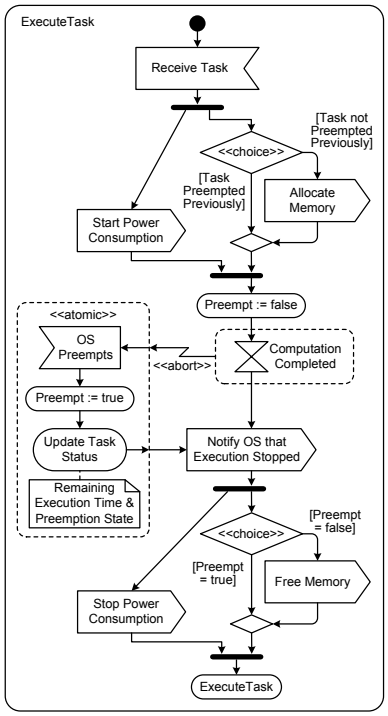

```
ScheduleTask()() |NewTask, OldTask: Task|

sel
   Tasks?Execute(NewTask)
      {TaskList register(NewTask)};
   if TaskList nextTask == NewTask then
      sel
         Processor!Execute(NewTask)
      or
         Processor!Preempt;
         Processor?Stopped(OldTask)
            {TaskList updateState(OldTask)};
         Processor!Execute(NewTask)
      les
   fi
or
   Processor?Stopped(OldTask)
      {TaskList remove(OldTask)};
   Tasks!ExecutionCompleted(OldTask);
   if TaskList notEmpty then
      Processor!Execute(TaskList nextTask)
   fi
les;
ScheduleTask()().
```

Fig. 12. POOSL Modeling pattern for preemptable scheduling of computations by a `Scheduler`

resources are interconnected, not even the topology of routers for a network-on-chip must be specified. These are seen as a result of the dependencies between application tasks and the chosen mapping.

3.3.3 *Mapping.* Figure 10 clearly shows that the Y-chart variant of SHE requires explicit specification of which processor resources execute what computations and which communication resources realize what communications [Wijk et al. 2003]. Explicitly mapping communications enables to abstract from the physical structure of how resources in the platform are interconnected, which conforms to the focuses on the service relations in Figure 2. By assuming a single interconnect between processor resources (i.e., a network-on-chip), the PREMADONA tool can automatically derive the mapping of communications from a mapping specified for computations. In that case, specifying the mapping of communications is obsolete [Theelen 2008].

The mapping of computations to processor resources and communications to communication resources is accomplished by means of exchanging service request - service acknowledgment messages between the involved modeling patterns, see also Figure 11. The fact that an application model gets feedback on what is going on in the platform is essential to model QoS and resource management as well as the reaction of the system to unpredictable events like user interactions [Goldschmidt and Hennessy 1993]. This approach differs from traditional trace-based mapping




```
ExecuteTask()() |StartTime: Real,
Preempt: Boolean, Task: Task|

OS?Execute(Task){StartTime := currentTime};
par
   Power!StartConsumption(ProcessorPower)
and
   if Task neverPreempted then
      Memory!Allocate(Task getRequiredMemory)
   fi
rap;
Preempt := false;
abort
   delay Task getExecutionTime
with
   OS?Preempt{Preempt := true;
      Task reduceTime(currentTime - StartTime)};
OS!Stopped(Task);
par
   Power!StopConsumption(ProcessorPower)
and
   if Preempt not then
      Memory!Free(Task getRequiredMemory)
   fi
rap;
ExecuteTask()().
```

Fig. 13. POOSL modeling pattern for preemptive execution of computations by a `Processor`

strategies as used in for example [Lieverse et al. 2001; Kienhuis et al. 2000].

3.3.4 *Evaluation (Analysis and Exploration).* SHE offers a broad spectrum of analysis opportunities based on the formal semantics of POOSL [Theelen et al. 2007]. The theory supports model checking by specifying functional correctness properties as well as best/worst-case, average-case and (expected/probabilistic) reachability performance properties in real-time temporal logics such as MTL [Koymans 1990], MITL [Alur 1991] and Temporal Rewards [Voeten 2002] using tools like SPIN [Holzmann 1991], UPPAAL [Larsen 1997] and PRISM [Kwiatkowska et al. 2002]. In case model checking is expected to suffer too much from state-space explosion, SHE offers simulation-based analysis of the mentioned property types as an alternative based on requiring explicitly extending a model with monitors expressed in POOSL. Predefined monitors for evaluating common types of long-run average metrics include accuracy analysis based on confidence intervals [Theelen 2004], where the estimation results are proven to converge to exactly the same results obtainable with model checking. SHE focuses on the evaluation of individual design alternatives. However, feeding the performance results obtained from models into approaches like those exploited in [Noonan and Flanagan 2006; Gries 2004; Thompson et al. 2007] would facilitate fully automated design-space exploration.

The PREMADONA tool utilises the simulation-based analysis techniques of SHE to enable evaluating various performance metrics as specified by the user in XML, see [Theelen 2008]. PREMADONA can add monitors to an application model for evaluat-

ing throughput, latency (minimum, maximum, average, variance), response delay, buffer occupancy (maximum, average, variance) and deadline miss probabilities. For platforms, it can add monitors to evaluate for example processor utilisation, memory occupancy (maximum, average, variance), communication load (maximum, average, variance) and power consumption (peak, average). PREMADONA could easily be extended to add monitors for evaluating any other metric of interest.

4. COMPARISON

This section compares the various design decisions and approaches used for implementing the methodologies discussed in the previous Sections. Table II[1][2][3] lists the features of Metropolis, DOL-MPA and the Y-chart variant of SHE.

4.1 Abstraction and Refinement

An important aspect of developing models during design is the necessity to abstract from implementation details [Theelen 2004]. Although literature mostly emphasises the advantage of increasing analysis efficiency, the key advantage of abstraction is actually the ability to focus on answering specific design questions. In case of the Y-chart paradigm, the focus is on performance-related questions and hence, one should abstract implementation details that do not affect performance to a great extend. Another reason for abstraction is the fact that the system is actually being designed and hence the implementation details are still to be decided.

The inherent difficulty of model-driven engineering is to make good abstractions such that a model properly represents the system, while still being able to answer the design questions of interest. On one hand, abstraction urges discarding many implementation details, while on the other hand, obtaining credible analysis results requires including those aspects that impact the performance (in our case). Models that properly capture all relevant aspects affecting the performance are sometimes called *adequate* [Theelen 2004]. Notice that adequacy is a property of a model, while *accuracy* is a property of an analysis result. The accuracy of a result depends highly on the type of analysis technique that is used for deriving the result; approximation, estimation or heuristic approaches will give less accurate results than exact techniques. Notice that adequacy of a model and accuracy of results are two orthogonal concepts. A 100% adequate model can give very inaccurate results for example due to combining it with inappropriate analysis techniques or by relying on simulations that ran way too short for the modeled behaviour to stabilise in the operation mode of interest. Conversely, a model can be very inadequate while analysing it gives 100% accurate results by using exact analysis techniques. Any model-driven engineering exercise includes a point in time where a constructed model must be considered as being adequate such that the analysis phase can start.

[1]Although Metropolis supports any behavior, resource, scheduling to be modeled in MMM, the table lists only those aspects for which elements are available in the Meta-Model library.
[2]Although DOL supports any behaviour to be specified, the table lists only features relevant to modeling and analysing systems with MPA
[3]Although SHE supports any behavior to be modeled in POOSL, the table lists only those aspects for which modeling patterns have been developed. The table also lists only those performance metrics for which the PREMADONA tool allows automatic addition of monitors to a model.

		Metropolis	*DOL-MPA*	*Y-chart Variant of SHE*
Application	*Supported MoCs*	YAPI, TTL and multi-rate synchronous dataflow	Task Graphs for Real-Time Systems	SDF, CSDF, SADF and Task Graphs for Real-Time Systems
Platform	*Supported Resources*	n/a	Processor, Communication	Processor, Communication, Storage, Energy
	Supported Schedulers	n/a	Preemptive and non-preemptive Fixed Priority and Rate Monotonic, preemptive Earliest Deadline First, First-Come First-Serve, Generalised Processor Sharing, Time Division Multiple Access and any hierarchical combinations of these	Preemptive and non-preemptive policies, including Round Robin (with or without skipping), Earliest Deadline First, First-Come First-Serve, Rate Monotonic, Generalised Processor Sharing and Time Division Multiple Access
	Refinement	Model element substitution	Model refinement with Real-Time Interfaces which ensure preservation of non-functional properties such as timing	SHE includes a model refinement approach ensuring preservation of functionality and timing, which is not used by the Y-chart variant
	Interconnection Type	Explicit platform interconnection structure	Implicit platform interconnection structure	Implicit platform interconnection structure
Mapping	*Approach*	LTL expressions extended with value mapping between application and platform events	Binding specification which defines mappings between processes and processor, software channels and communication paths. Resource streams are inputs for performance components	Service request - acknowledgement interactions between application and platform models
	Mapped Elements	Computations only	Computations and communications	Computations and communications
Analysis	*Application*	Throughput, end-to-end delays, buffer occupancy	Throughput, end-to-end delays, buffer occupancy	Throughput, response delay, inter-firing latency, buffer occupancy, deadline miss probabilities
	Platform	Processor load, utilization of communication resources, memory occupancy	Processor load, utilization of communication resources, memory occupancy, power consumption	Processor load, utilization of communication resources, memory occupancy, power consumption
	Type	Simulation: Worst, best and average case. LTL and LOC monitors. Model checking : LTL via SPIN LOC via Checker	Worst and best case	Simulation-based analysis of functional correctness and worst/best-case, average case and probabilistic/expected reachability performance metrics. SHE has model-checking techniques for these metrics but there are no automated tools
	Accuracy	All results from model checking are exact. All results from simulation are approximations.	Hard upper and lower bounds	Observation results for functional correctness and worst/best case and probabilistic/expected reachability performance metrics. Confidence intervals with accuracy result for average-case performance metrics
Exploration	*Type*	Manual exploration via alternative platforms and alternative mappings	Use of DOL; Manual exploration or automatic with an optimization framework such as [Bleuler et al. 2003] based on multi-objective evolutionary algorithms	Manual exploration or the approach of [Noonan and Flanagan 2006] based on evolutionary algorithms

Table II. A comparison of Y-chart based design methodologies

All three methodologies in Table II focus on the evaluation of performance properties and therefore prescribe or require abstraction from functional details and the actual content of data that is being processed. Both Metropolis and SHE would in principle allow refining these aspects to complete implementation details, but

they encourage a designer to capture only those aspects that (potentially) affect the performance. By relying on the Real-Time Calculus, MPA goes even a step further in disabling the possibility of specifying implementation details completely. As a consequence, developing adequate models in MPA may be more difficult than when using Metropolis or SHE but the advantage is a better analysis efficiency and scalability. Conversely, all three methodologies support model refinement towards a more detailed specifications. Both SHE and MPA include formal techniques for model refinement by means of decomposing model components into a collection of more detailed components. Metropolis also supports such model refinement but only certain approaches can guarantee that properties don't change by refinement. SHE furthermore includes formal techniques for synthesising real-time control software on single-processor platforms that guarantee preservation of functionality and performance as specified and analyzed in a model. This work needs however further extension to allow application in the MPSoC setting of the Y-chart paradigm.

Popular abstraction mechanisms are the use of probabilistic distributions and non-determinism. Both these mechanisms allow abstract specification of choices. As opposed to non-determinism, using stochasticity requires knowledge about the relative occurrence of each possible alternative. The ability to use these mechanisms strongly depends on the formalisms underlying a methodology (see also Table I). Both Metropolis and SHE support the use of non-deterministic choices between alternative behaviours or alternative data items. Metropolis offers the use of non-determinate variables to express the latter. However, such variables become concrete data items when interpreting a mapping specification for a system. POOSL allows non-determinism between alternative data values by using different assignments (possibly of different types of data) in a non-deterministic behavioural choice. Non-deterministic behaviour is supported by a specific language primitive in POOSL, which concerns a selection between alternative actions. MMM includes a similar language primitive. There is however also a difference between the non-deterministic choice in MMM and POOSL. In case no alternatives are enabled for the non-deterministic choice in MMM, the overall behaviour will block forever. In POOSL, a similar situation may occur but other behaviour running in parallel with the blocked non-deterministic behaviour may unblock one or more of the alternatives. For MPA, non-determinism is inherently present in the arrival/service curves due to abstracting from the exact moments in time that data is communicated.

The use of probabilities is only supported in the SHE methodology. The SADF application models accepted by the PREMADONA tool are an example of where probabilistic choices between alternative behaviour and timing can be specified.

4.2 Explicit versus Implicit Resource Interconnects

Table II states that Metropolis, DOL-MPA and the Y-chart variant of SHE use different ways to specify the topology of platform resources as shown in Figure 14.

Platform models in Metropolis have an explicit resource topology, see top half of Figure 14. Hence, the models explicitly capture how for example processor resources are interconnected via communication resources. Because of this explicit topology, the mapping specification only defines the correspondences between computations in the application model and processors in the platform model. The path that messages take between computations is implicitly defined by the chain

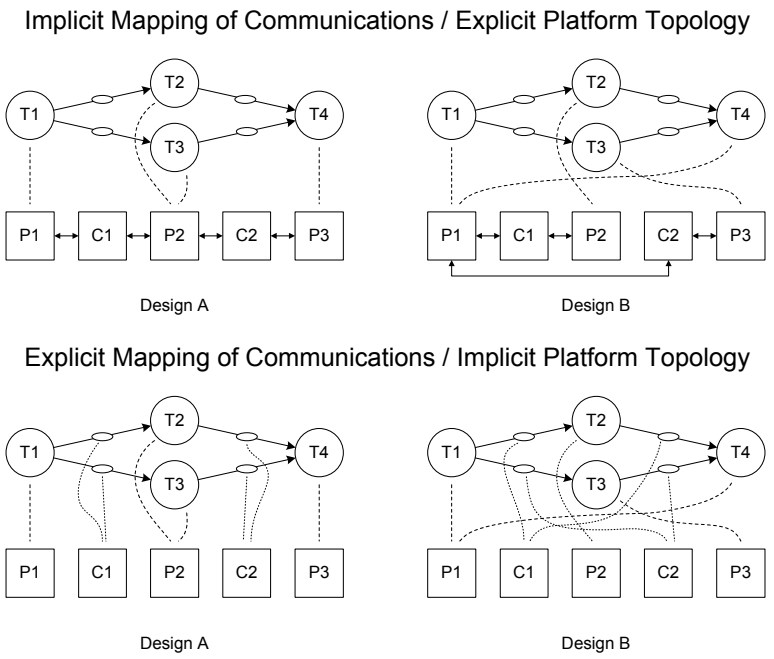


Fig. 14. Approaches to capture mapping and the topology of resources in platforms

of communication resources between the processors that execute the computations and the mapping. This approach has the benefit that it is easy to define complex (hierarchal) combinations of communication or storage resources, where busses or memories serve other busses and memories respectively, while having a small mapping specification. The disadvantage is that changing the mapping may require to change the platform model as well in order to ensure that resource interconnections are consistent with dependencies between computations and the chosen mapping.

On the other hand, the Y-chart variant of SHE uses platform models that do not contain any topological information. This approach is illustrated in the lower half of Figure 14. The mapping specification contains a binding for both computations and communications of an application. Hence, the topology of how the processor and communication resources are interconnected emerges from the mapping specification. This gives much flexibility when evaluating alternative mappings since there is no need to change the platform model to ensure that resource interconnections are consistent with dependencies between computations and the chosen mapping. However, in case complex (hierarchical) combinations of communication or storage resources are to be considered, one would need to introduce artificial elements in the application model that are not part of the real application. These would represent the tasks of bridging protocols between communication resources or the transfer of data between two storage resources respectively. The reason for the need to introduce these artifacts in the application model is that the mapping rules do not allow distributing for example a communication in the application over multiple communication resources such as a hierarchy of busses.

Platform models in the context of DOL-MPA are closer to the approach taken by Metropolis. Platform models define explicitly the resources which are available such as processors, memories and their interconnections. Moreover, end-to-end communication paths with nodes on the affected resources are used in order to model communication. This allows networks-on-chip to be modeled adequately. As discussed earlier, mapping must be done in the spatial domain as well as in the temporal domain. The spatial domain mapping specification explicitly defines the binding between processes and software channels to their corresponding processors and communication paths. This is very similar to Metropolis but with the extra task of mapping the communication paths. The approach has the advantage of explicitly stating which communication path should be used between two resources when multiple paths are present. This facilitates end-to-end communication performance analysis. The temporal mapping specification explicitly defines the scheduling policy on each resource and the corresponding parameters. DOL-MPA differs from Metropolis and the Y-chart variant of SHE in that scheduling is part of the mapping and not part of the platform model.

4.3 Exact Analysis versus Simulation-Based Analysis

According to Table II, Metropolis, DOL-MPA and the Y-chart variant of SHE accomplish the fourth step of the Y-chart paradigm (See Section 2.1) on evaluating a proposed design solution in different ways. The considered approaches can be categorised in exact analysis and simulation-based analysis. Metropolis and SHE offer state-space exploration based analysis techniques for correctness and performance properties, while DOL-MPA supports exact analysis based on the Real-Time Calculus [Thiele et al. 2000; Chakraborty et al. 2003]. Metropolis utilises traditional model checking techniques for determining whether LTL and LOC properties are satisfied. For exact analysis of functional correctness, SHE follows the approach of linear-time temporal logic verification, where typically the logical negation of a required property is converted into an automaton by means of a so-called tableau construction [Geilen 2001]. Subsequently, automata theoretic techniques are used to test whether the property is satisfied. Although the properties that can be model checked include certain types of timing properties, the traditional approach is not suitable for amongst others long-run average performance metrics like throughput. To compute exact results for such metrics, SHE supports a more liberal form of model checking that relies on Markov theory [Theelen et al. 2007; Theelen 2004]. Nevertheless, SHE does not include automatic tools to actually perform these calculations as opposed to Metropolis. An important disadvantage of state-space based analysis is that it does not scale to large systems. MPA circumvents this problem by using Real-Time Calculus as an alternative exact analysis approach. The Real-Time Calculus is derived from the Max-Plus algebra [Baccelli et al. 1992], which allows computing the latest moment in time at which events (e.g., the arrival of data) can occur. Using this as a basis, analysis with MPA is limited to determining performance bounds.

Since Metropolis and SHE suffer from state-space explosion issues when using their exact analysis techniques, they offer simulation-based analysis as a scalable alternative. However, simulations are never exhaustive (in general) and the obtained results are only valid for that part of the state-space that is actually covered

during a simulation. Hence, simulation-based analysis gives estimations of the actual performance of a system. An essential problem is therefore how long a simulation should run before the results are considered accurate. While Metropolis does not provide support to evaluate the accuracy of results, SHE includes techniques to evaluate the accuracy of results for any type of long-run average performance metric based on confidence intervals [Theelen 2004]. Another crucial feature of SHE is that the simulation-based estimation results converge to the same results that would be obtainable with its exact techniques. This originates from founding both approaches on the same mathematical model defined by the formal semantics of POOSL. Informal simulation-based approaches (including those for Metropolis) cannot guarantee such correspondence between exact and estimation results.

5. CONCLUSION AND FUTURE TRENDS

Embedded systems become more complex with each generation. A good portion of this complexity originates from the growing level of heterogeneity. Modern systems include a variety of components types that are not electronic such as micro-mechanical and micro-optical. In addition, there is a certain momentum in the industry toward IP based design. In order to adequately support these tendencies, implementations of the Y-chart paradigm will have to support MoCs and MoAs with sufficient expressivity. New MoCs or adaptations of existing ones will have to be developed. A long-standing challenge in this area are MoCs supporting the integration of various abstraction levels. On the other hand, MoAs are currently fairly immature - most are adaptations of existing MoCs. Hence, developing sufficiently expressive MoAs is an interesting challenge for future research.

Application of the Y-chart paradigm to domains other than streaming multimedia systems is a subject of current research. The domain of high-tech mechatronic systems is a good example that could benefit for the Y-chart paradigm. Problems that originally arose in streaming multi-media systems are now also observed in these types of systems, where feedback and feedforward control strategies have to be implemented with tight performance requirements.

Most current Y-chart based design methodologies focus on the tasks of modeling and analysis. However, in order to meet the objective of shortening time-to-market, a new generation of methodologies are required, which incorporate extensive synthesis capabilities. The main challenge to address is the issue of heterogeneity. Analysis models are of a different nature than synthesis models. They are optimised for efficient determination of properties of interest and therefore only contain the essential information. The focus of synthesis is efficient implementation. Synthesis models are therefore more detailed, but they also need to be complete. In general analysis models cannot simply be refined into synthesis models, making it difficult to establish proper relations between them. Understanding these relations is a prerequisite to integrate them into seamless design flows.

This article explores the richness of how design methodologies have implemented the Y-chart paradigm . The Y-chart paradigm was developed more then 10 years ago in order to address the challenges surrounding the exploration of the design-space of streaming systems. It is defined by a simple Y-shaped sequence of tasks (functional application modeling, platform architecture modeling, mapping, evalu-

ation and synthesis). The philosophy of separating functional design and platform design offers a very effective solution to design problems, where alternative deployments of platforms are an essential aspect. Despite the simplicity and straightforwardness of the Y-chart paradigm, many variations exist in implementing it in design methodologies. Key areas of differences are (i) the supported MoCs and MoAs, (ii) the approaches used for mapping, and (iii) the type of metrics that can be analyzed as well as the level of accuracy of results.

The comparison has highlighted that each design approach has its advantages and inconveniences. Metropolis has the advantage of offering a simple yet very powerful mapping approach by using LTL expressions. Moreover, by using LTL expressions, application and platform models are truly kept separate. A relative weakness of Metropolis vs the other implementations is the informal nature of models which limits evaluation to mostly simulation. The Y-Chart variant of SHE/POOSL offers a great deal of expressivity due to the POOSL language and simulation analysis because of SHE. Moreover, integration with other design paradigms it possible because the implementation of the Y-Chart approach is achieved in the context of a broader methodology framework (SHE). A relative weakness of the implementation is that the application models must make explicit calls to the platforms models which bind them. Moreover, the implicit platform topologies of the platform because harder to use for complex cases. DOL-MPA has the advantage of offering exact analysis through the means of Real-Time Calculus on models with no implementation details. However, the use of Real-time Calculus has the disadvantage of being harder to use.

REFERENCES

Alur, R. 1991. Techniques for Automatic Verification of Real-Time Systems. Ph.D. thesis, Stanford University, California, USA.

Baccelli, F., Cohen, G., Olsder, G., and J.P.Quadrat. 1992. *Synchronization and Linearity.* John Wiley & Sons.

Balarin, F., Burch, J., Lavagno, L., Passerone, R., Sangiovanni-Vincentell, A., and Watanabe, Y. 2001. Constraints Specification at Higher Levels of Abstraction. In *Proceedings of HLDVT.* IEEE, 129–133.

Balarin, F., Watanabe, Y., Hsieh, H., Lavagno, L., Passerone, C., and Sangiovanni-Vincentelli, A. 2003. Metropolis: An Integrated Electronic Design Environment. *IEEE Computer 36,* 4, 45–52.

Balarin, F., G. P. J. A. P. C. S. E. T. B. C. M. H. H. L. L. S.-V. A. S. K. 1997. *Hardware-Software Co-Design of Embedded Systems: The POLIS Approach.* Springer.

Baleani, M., Ferrari, A., Sangiovanni-Vincentelli, A., and Turchetti, C. 2000. HW/SW Codesign of an Engine Management System. In *Proceedings of DATE.* IEEE, 263–267.

Bezivin, J. 2005. On the Unification Power of Models. *Software and System Modeling 4,* 2, 171–188.

Bleuler, S., Laumanns, M., Thiele, L., and Zitzler, E. 2003. PISA - A Platform and Programming Language Independent Interface for Search Algorithms. In *Evolutionary Multi-Criterion Optimization (EMO 2003).* Springer Verlag, Faro, Portugal, 494 – 508.

Bokhoven, L. v. 2002. Constructive Tool Design for Formal Languages: From Semantics to Executing Models. Ph.D. thesis, Eindhoven University of Technology, Eindhoven, Netherlands.

Brand, D. and Zafiropulo, P. 1983. On Communicating Finite-State Machines. *Journal of the ACM 20,* 2, 323–342.

Brooks, C., Lee, E., Liu, X., Neuendorffer, S., Zhao, Y., and Zheng, H. 2005. Heterogeneous

Concurrent Modeling and Design in Java (Volume 1: Introduction to Ptolemy II). Tech. Rep. UCB/ERL M05/21, University of California, Berkeley, USA.

Burch, J., Passerone, R., and Sangiovanni-Vincentelli, A. 2001. Overcoming Heterophobia: Modeling Concurrency in Heterogeneous Systems. In *Proceedings of ACSD*. IEEE, 13–32.

Cesario, W., Nicolescu, G., Gauthier, L., Lyonnard, D., and Jerraya, A. 2001. Colif: A Design Representation for Application-Specific Multiprocessor SoCs. *Design & Test of Computers 18,* 5, 8–20.

Chakraborty, S., Künzli, S., and Thiele, L. 2003. A General Framework for Analysing System Properties in Platform-Based Embedded System Designs. In *Design Automation and Test in Europe (DATE)*. IEEE Press, Munich, Germany, 190–195.

Chakraborty, S., Künzli, S., Thiele, L., Herkersdorf, A., and Sagmeister, P. 2003. Performance Evaluation of Network Processor Architectures: Combining Simulation with Analytical Estimation. *Computer Networks 41,* 5 (April), 641–665.

Chakraborty, S., Mitra, T., Roychoudhury, A., Thiele, L., Bordoloi, U., and Derdiyok, C. 2007. Cache-Aware Timing Analysis of Streaming Applications. In *Proceedings of the 19th Euromicro Conference on Real-Time Systems*. IEEE Computer Society Washington, DC, USA, 159–168.

Chen, X., Davare, A., Hsieh, H., Sangiovanni-Vincentelli, A., and Watanabe, Y. 2005. Simulation based deadlock analysis for system level designs. In *Proceedings of DAC*. IEEE, 260–265.

Cruz, R. 1991. A Calculus for Network Delay, Part I: Network Elements in Isolation. *IEEE Transactions on Information Theory 37,* 1, 114–131.

Davare, A., Densmore, D., Meyerowitz, T., Pinto, A., Sangiovanni-Vincentelli, A., Yang, G., Zeng, H., and Zhu, Q. 2007. A Next-Generation Design Framework for Platform-Based Design. In *Proceedings of DVCon*. IEEE.

Densmore, D. 2004a. Formal Refinement Verification in Metropolis. Tech. rep., University of California, Berkeley, USA. UCB/ERL M04/10.

Densmore, D. 2004b. Metropolis Architecture Refinement Styles and Methodology. Tech. rep., University of California, Berkeley, USA. UCB/ERL M04/36.

Densmore, D., Passerone, R., and Sangiovanni-Vincentelli, A. 2006. A Platform-Based Taxonomy for ESL Design. *IEEE Design & Test of Computers 23,* 5, 359–374.

Densmore, D., Rekhi, S., and Sangiovanni-Vincentelli, A. 2004. Microarchitecture Development via Metropolis Successive Platform Refinement. In *Proceedings of DATE*. IEEE, 10346–10357.

Florescu, O. 2007. Predictable Design for Real-Time Systems. Ph.D. thesis, Eindhoven University of Technology, Eindhoven, Netherlands.

Florescu, O., Voeten, J., Verhoef, M., and Corporaal, H. 2007. Reusing Real-Time Systems Design Experience through Modeling Patterns. In *Advances in Design and Specification Languages for Embedded Systems*, S. Huss, Ed. Springer, Chapter 20, 329–348.

Gajski, D. and Kuhn, R. H. 1983. New VLSI Tools. *Computer*, 11–14.

Gajski, D., Shu, J., Raider, D., Gerstlauer, A., and Zhao, S. 2000. *SpecC: Specification Language and Methodology*. Springer.

Geilen, M. 2001. On the Construction of Monitors for Temporal Logic Properties. In *Proceedings of the 1st Workshop on Runtime Verification (RV'01)*.

Geilen, M. 2002. Formal Techniques for Verification of Complex Real-Time Systems. Ph.D. thesis, Eindhoven University of Technology, Eindhoven, Netherlands.

Ghamarian, A. 2008. Timing Analysis of Synchronous Data Flow Graphs. Ph.D. thesis, Eindhoven University of Technology, Eindhoven, Netherlands.

Goessler, G. and Sangiovanni-Vincentelli, A. 2002. Compositional Modeling in Metropolis. In *Proceedings of EMSOFT*. Springer, 93–107.

Goldschmidt, S. and Hennessy, J. 1993. The Accuracy of Trace-Driven Simulations of Multiprocessors. *Performance Evaluation Review 21,* 1, 146–157.

Gries, M. 2004. Methods for Evaluating and Covering the Design Space during Early Design Development. *Integration, the VLSI Journal 38,* 2, 131–183.

Gries, M. and Keutzer, K. 2005. *Building ASIPs: The Mescal Methodology*. Springer.

Grotker, T., Liao, S., Martin, G., and Swan, S. 2002. *System Design with SystemC*. Kluwer Academic Publications.

H. Chang, L. Cooke, H. H. G. M. A. M. and Todd, L. 1999. *Surviving the SOC Revolution : A Guide to Platfrom-Based Design*. Klumer Academic.

Hamann, A., Jersak, M., Richter, K., and Ernst, R. 2004. Design Space Exploration and System Optimization with SymTA/S - Symbolic Timing Analysis for Systems. In *Proceedings of RTSS*. IEEE, 469–478.

Holzmann, G. 1991. *Design and Validation of Computer Protocols*. Prentice-Hall.

Huang, J. 2005. Predictability in Real-Time System Design. Ph.D. thesis, Eindhoven University of Technology, Eindhoven, Netherlands.

Kienhuis, B. 1999. Design Space Exploration of Stream-based Dataflow Architectures: Methods and Tools. Ph.D. thesis, Delft University of Technology, Delft, Netherlands.

Kienhuis, B., Deprettere, E., Vissers, K., and Wolf, P. v. 1997. An Approach for Quantitative Analysis of Application-Specific Dataflow Architectures. In *Proceedinsg of ASAP*. IEEE, 338–349.

Kienhuis, B., Depretterel, E., Vissers, K., and van der Wolf, P. 1997. An Approach for Quantitative Analysis of Application-Specific Dataflow Architectures. *ASAP*, 338–349.

Kienhuis, B., Rijpkema, E., and Deprettere, E. 2000. Compaan: Deriving Process Networks from Matlab for Embedded Signal Processing Architectures. In *Proceedings of CODES*. IEEE, 13–17.

Kock, E. d., Essink, G., Smits, W., Wolf, P. v., Brunel, J., Kruijtzer, W., Lieverse, P., and Vissers, K. 2000. YAPI: Application Modeling for Signal Processing Systems. In *Proceedings of DAC*. ACM, 402–405.

Koymans, K. 1990. Specifying Real-Time Properties with the Metric Temporal Language. *Real-Time Systems 2,* 4, 255–299.

Krolikoski, S. J., Schirrmeister, F., Salefski, B., Rowson, J., and Martin, G. 1999. Methodology and Technology for Virtual Component Driven Hardware/Software Co-Design on the System-Level. *ISCAS*, 456–459.

Kruijtzer, W., Wolf, P., Kock, E., Stuyt, J., Ecker, W., Maye, A., Hustin, S., Amerijckx, C., Paoliand, S., and Vaumorin, E. 2008. Industrial IP Integration Flows based on IP-XACT Standards. In *Proceedings of DATE*. IEEE, 26–31.

Kumar, A., Mesman, B., Theelen, B., and Corporaal, H. 2006. Resource Manager for Non-Preemptive Heterogeneous Multiprocessor System-on-Chip. In *Proceedings of ESTIMedia*. IEEE, 33–38.

Kwiatkowska, M., Norman, G., Segala, R., and Sprontston, J. 2002. Automatic Verification of Real-Time Systems with Discrete Probability Distributions. *Theoretical Computer Science 282,* 1, 101–150.

Larsen, K. 1997. UPPALL in a Nutshell. *Journal on Software Tools for Technology Transfer 1,* 1-2, 134–152.

Lee, E. and Messerschmitt, D. 1987. Synchronous Data Flow. *IEEE Proceedings 75,* 9, 1235–1245.

Lee, E. and Sangiovanni-Vincentelli, A. 1998. A framework for Comparing Models of Computation. *Transactions on Computer-Aided Design of Integrated Circuits and Systems 17,* 12, 1217–1229.

Lieverse, P., Wolf, P. v. d., Vissers, K., and Deprettere, E. 2001. A Methodology for Architecture Exploration of Heterogeneous Signal Processing Systems. *Journal of VLSI Signal Processing Systems for Signal, Image and Video Technology 29,* 3, 197–207.

Martin, G. 1998. Design Methodologies for System Level IP. In *Proceedings of DATE*. IEEE, 286–289.

Medvidovic, N. and Taylor, R. 2000. A Classification and Comparison Framework for Software Architecture Descripton Languages. *Transactions on Software Engineering 26,* 1, 70–93.

Meyerowitz, T. 2004. Metropolis ARM CPU Examples. Tech. Rep. UCB/ERL M04/39, University of California, Berkeley, USA.

Mishra, P. and Dutt, N. 2008. *Processor Description Languages: Applications and Methodologies*. Morgan Kaufmann.

Noonan, L. and Flanagan, C. 2006. Utilising Evolutionary Approaches and Object-Oriented Techniques for Design-Space Exploration. In *Proceedings of DSD*. IEEE, 346–352.

Pinto, A. 2004. Metropolis Design Guidelines. Tech. Rep. UCB/ERL M04/40, University of California, Berkeley, USA.

Putten, P. v. d. and Voeten, J. 1997. Specification of Reactive Hardware/Software Systems. Ph.D. thesis, Eindhoven University of Technology, Eindhoven, Netherlands.

Qin, W. and Malik, S. 2002. Architecture Description Languages for Retargetable Compilation. In *The Compiler Design Handbook: Optimizations & Machine Code Generation*. CRC Press, 535–562.

Sangiovanni-Vincentelli, A. and Martin, G. 2001. Platform-Based Design and Software Design Methodology for Embedded Systems. *Design and Test of Computers 18,* 6, 21–33.

Sgroi, M., Sheets, M., Mihal, A., Keutser, K., Malik, S., Rabaey, J., and Sangiovanni-Vincentelli, A. 2001. Adressing the System-on-a-Chip Interconnect Woes Through Communication-Based Design. In *Proceedings of DAC*. IEEE, 667–672.

Stuijk, S. 2007. Predictable Mapping of Streaming Applications on Multiprocessors. Ph.D. thesis, Eindhoven University of Technology, Eindhoven, Netherlands.

Teich, J., Blickle, T., and Thiele, L. 1998. System-Level Synthesis Using Evolutionary Algorithms. *J. Design Automation for Embedded Systems 3*, 23–58.

Theelen, B. 2004. Performance Modelling for System-Level Design. Ph.D. thesis, Eindhoven University of Technology, Eindhoven, Netherlands.

Theelen, B. 2007. A Performance Analysis Tool for Scenario-Aware Streaming Applications. In *Proceedings of QEST*. IEEE, 269–270.

Theelen, B. 2008. Performance Model Generation for MPSoC Design-Space Exploration. In *Proceedings of QEST*. IEEE.

Theelen, B., Florescu, O., Geilen, M., Huang, J., Putten, P. v., and Voeten, J. 2007. Software/Hardware Engineering with the Parallel Object-Oriented Specification Language. In *Proceedings of MEMOCODE*. IEEE, 139–148.

Thiele, L., Bacivarov, I., Haid, W., and Huang, K. 2007. Mapping Applications to Tiled Multiprocessor Embedded Systems. In *Proc. 7th Intl Conference on Application of Concurrency to System Design (ACSD 2007)*. IEEE Computer Society, Bratislava, Slovak Republic, 29–40.

Thiele, L., Chakraborty, S., and Naedele, M. 2000. Real-time Calculus for Scheduling Hard Real-time Systems. In *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*. Geneva, Switzerland, 101–104.

Thiele, L., Wandeler, E., and Stoimenov, N. 2006. Real-Time Interfaces for Composing Real-Time Systems. In *International Conference On Embedded Software (EMSOFT 06)*. Seoul, Korea, 34–43.

Thompson, M., Nikolov, H., Stefanov, T., Pimentel, A., Erbas, C., Polstra, S., and Deprettere, E. 2007. A Framework for Rapid System-Level Exploration, Synthesis, and Programming of Multimedia MP-SoCs. In *Proceedings of CODES/ISSS*. IEEE, 139–148.

Voeten, J. 2002. Performance Evaluation with Temporal Rewards. *Performance Evaluation 50,* 2/3, 189–218.

Wandeler, E. and Thiele, L. 2006a. Interface-Based Design of Real-Time Systems with Hierarchical Scheduling. In *12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. San Jose, USA, 243–252.

Wandeler, E. and Thiele, L. 2006b. Real-Time Calculus (RTC) Toolbox.

Wandeler, E. and Thiele, L. 2007. Workload Correlations in Multi-processor Hard Real-time Systems. *Journal of Computer and System Sciences 73,* 2 (March), 207–224.

Wandeler, E., Thiele, L., Verhoef, M., and Lieverse, P. 2006. System Architecture Evaluation using Modular Performance Analysis: A Case Study. *International Journal on Software Tools for Technology Transfer 19*, 649–667.

Wijk, F. V., Voeten, J., and Berg, A. t. 2003. An Abstract Modeling Approach Towards System-Level Design-Space Exploration. In *System Specification and Design Languages*, E. Villar and J. Mermet, Eds. Kluwer Academic Publishers, Chapter 22, 267–282.

Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D., Bernat, G., Ferdinand, C., Heckmann, R., Mueller, F., Puaut, I., Puschner, P., Staschulat, J., and Stenstrm, P. 2008. The Determination of Worst-Case Execution Times—Overview of the Methods and Survey of Tools. *ACM Transactions on Embedded Computing Systems (TECS) 7,* 3 (April).

Yang, G., Hsieh, H., Chen, X., Balarin, F., and Sangiovanni-Vincentelli, A. 2006. Constraints Assisted Modeling and Validation in Metropolis Framework. In *Proceedings of Asilomar Conference on Signal, Systems and Computers*. IEEE, 1469–1474.

Zeng, H., Davare, A., Sangiovanni-Vincentelli, A., Sonalkar, S., Kanajan, S., and Pinello, C. 2006. Design SPace Exploration of Automotive Platforms in Metropolis. In *Scociety of Automotive Engineers Congress*.

Received Month Year; Revised Month Year; accepted Month Year