The Artemis Architecture Workbench

A.D. Pimentel[†] P. van der Wolf[‡] E.F. Deprettere[§] L.O. Hertzberger[†]
J. T. J. van Eijndhoven[‡] S. Vassiliadis[◇]

† Computer Architecture & Parallel Systems group, University of Amsterdam
‡ Philips Research Laboratories, Eindhoven
§ Leiden Institute of Advanced Computer Science, Leiden University
◇ Computer Engineering group, Delft University of Technology

Abstract—**Modern signal processing and multimedia embedded systems increasingly need to be able to support a wide range of applications and standards. These systems require programmable components for their implementation, whereas reconfigurable hardware components can also be used for time-critical tasks without compromising flexibility. The heterogeneity of such embedded systems and the different demands of their target applications greatly complicate the system design. There are currently no mature tools available addressing these new aspects.**

In this paper, we provide an overview of the Artemis project which aims at the development of methods and techniques to support the design of highly programmable embedded media systems. The result of Artemis will be a simulation environment for system architecture design space exploration as well as an experimentation framework for reconfigurable architectures. The combination of these two frameworks provides a workbench for identifying computationally intensive tasks in applications which are suitable for execution on reconfigurable hardware components. The simulation workbench will provide support for evaluating instantiations of embedded systems architectures at multiple levels of abstraction. For this purpose, the level of detail of architecture models will be adjustable in a seamless manner, while also facilitating mixed-level architecture simulations.

Keywords— **Highly-programmable embedded systems, media and signal processing, system architecture design, reconfigurable architectures, design space exploration, computer architecture simulation**

I. Introduction

Modern embedded systems, like those for media and signal processing, increasingly need to be multifunctional and must support multiple standards. A high degree of *programmability*, which can be provided by applying microprocessor technology as well as reconfigurable hardware, is key to the development of such advanced embedded systems. Due to the combination of programmable, reconfigurable and dedicated hardware components, we refer to these systems as *heterogeneous embedded systems*. With the emergence of highly programmable heterogeneous embedded systems, we have to reconsider the suitability of the existing techniques for their hardware/software codesign [1]. Particularly, the traditional application-driven approach in which a certain application is gradually synthesized into the appropriate hardware and software components is not suited for the design of these programmable systems. Such an approach is ideal for the design of dedicated systems, but it lacks the generalizability to cope with programmable architectures suited for the processing of a broad range of applications.

Another trend which affects the design of embedded systems is the need for reducing the design time to allow a short time-to-market. At the same time, the complexity of the designs is increasing. The reason for this is twofold. Firstly, the systems have to satisfy the performance requirements of a range of applications. Secondly, technology improvements simply allow for more and more functionality to be integrated on a single chip. Reducing the design time of these increasingly complex embedded systems implies that the traditional approach of relying on detailed simulation models, such as provided by cycle-true simulators, as the most important vehicle for the design space exploration has become infeasible. The effort required to build such detailed simulators is relatively high, making it impractical to use them in the very early design stages. Moreover, the low simulation speeds of these simulators significantly hamper the architectural exploration. To solve this problem, a co-design environment should allow for simulation at a range of abstraction levels. This way, the speed, required modeling effort and attainable accuracy of the architecture simulations can be explicitly controlled. In such a co-design environment, abstract simulation models are used for exploring the large design space in the early design stages, while in a later stage more low-level hardware models can be applied for functional verification or detailed performance studies. Hence, this calls for concepts to refine the simulation models across different abstraction levels in a smooth manner, and which also allow for mixed-level architecture simulations.

The Artemis (ARchitectures and meThods for Embedded MedIa Systems) project aims at reducing the design

time of embedded media systems with a high degree of programmability. To this end, a simulation workbench for architecture design space exploration as well as an experimentation framework for reconfigurable architectures are developed. In this paper, we present an overview of the Artemis project and indicate the research challenges that are ahead of us.

The remainder of the paper is organized as follows. The next section gives a general description of the Artemis project. Section III describes how Artemis relates to other efforts in the fields of simulation of embedded systems architectures and reconfigurable computing. In Section IV, we provide an overview of the architecture simulation workbench and explain how application models are mapped onto architecture models. Section V describes our research effort regarding reconfigurable architectures. Finally, in Section VI we draw our conclusions.

II. The Artemis project

To help reducing the design time of highly programmable embedded systems, the Artemis project addresses two research challenges. First, an architecture simulation workbench is being developed which provides methods, tools and libraries for the efficient exploration of heterogeneous embedded systems architectures. With the term *efficient*, we mean that the simulation workbench allows for rapidly evaluating different architecture designs, application-architecture mappings and hardware/software partitionings at various levels of abstraction and for a broad range of (media) applications. As we will explain, the key to this high degree of evaluation flexibility is the recognition of *separate application and architecture models* in the system simulation. The application model purely describes the functional behavior of an application while the architecture model mimics the timing behavior of the architecture onto which the application is mapped.

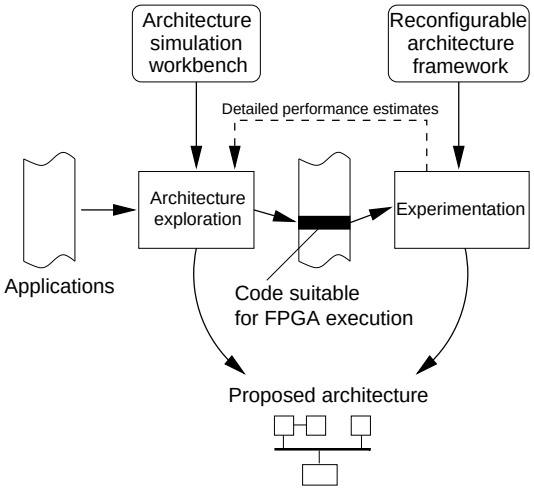

Fig. 1. The Artemis architecture workbench.

The second research challenge of Artemis consists of investigating the potentials of reconfigurable embedded computer architectures as a new means to enhance the programmability of embedded systems. Reconfigurable computing refers to an implementation style where the exact functional behavior of a piece of hardware can be configured in order to efficiently process a particular task, and can be reconfigured later for a different task. Reconfigurable hardware components, implemented using for example FPGAs, have the potential to deliver a high performance for specific applications with a limited power consumption, while retaining flexibility. The combination and integration of reconfigurable components with existing architectural components is a novel problem. In particular, integrating a reconfigurable component with a programmable core, like a VLIW processor, is the subject of our research.

Figure 1 illustrates how both research activities are integrated into the *Artemis architecture workbench* and how this workbench is used for the design and exploration of architectures for heterogeneous embedded media systems. The two central parts in Artemis are the architecture simulation workbench and the reconfigurable architecture framework. By applying the simulation workbench to a set of target applications, potential architecture designs can be swiftly evaluated. The architecture design space exploration performed at this stage results in recommendations on candidate heterogeneous architectures. In addition, analysis of application-architecture mappings (i.e. which application task is mapped onto which architecture component) is also used to select time-critical candidate tasks in applications for execution on reconfigurable components. These selected code fragments are used as input to the tools from the reconfigurable architecture framework to thoroughly study their mapping onto a reconfigurable component. Such a study will, on its turn, produce accurate performance estimates of the reconfigurable execution which can again be used for the validation and calibration of the models of the architecture simulation workbench. Evidently, the combination of the architecture simulation workbench and the reconfigurable architecture framework should in the end lead to a proposal for a heterogeneous system architecture which allows for efficient processing of the target applications.

III. Putting Artemis into perspective

System architecture simulation in the context of heterogeneous embedded systems is a relatively new research field which has received a lot of attention in recent years.

The magic word in most of the research and commercial efforts in this field is *co-simulation.* Like its name already suggests, co-simulation implies that the software parts (which will be mapped onto a programmable processor) and the hardware components of an embedded application and their interactions are simulated together in one simulation [2]. Current co-simulation frameworks typically combine two (rather low-level) simulators, one for simulating the programmable components running the software and one for the dedicated hardware. For software simulation, instruction-level processor simulators are often used, whereas HDLs such as VHDL and Verilog are usually used for the hardware simulation. Figure 2(a) illustrates the general idea of co-simulation. In this figure, the white half circles refer to the software components which are executed on programmable architecture components (black half circles) and the arrows indicate the interactions between the software and hardware simulators. The hardware and software simulators are either running apart from each other (e.g. the co-simulators in Seamless CVE, EagleI/EagleV, Virtual CPU and CoSim [3], Coumeri's [4] and Bauer's [5] environments and Symphony [6]) or they are integrated to form one monolithic simulator (e.g. Poseidon [7], Polis [8], Pia [9] and the work of Soininen et al. [10]). A major drawback of all of these co-simulators is their inflexibility. Because they all make an explicit distinction between software and hardware simulation, it must already be known which application components will be performed in software and which ones in hardware before the system model is built. This significantly complicates the evaluation of different hardware/software partitioning schemes since a whole new system model may be required for the assessment of each partitioning. For this reason, the co-simulation stage is often preceded by a stage in which the application is studied in isolation by means of a functional software model written in a high level language. This typically results in rough estimations of the application's performance requirements, which are subsequently used as guidance for the hardware/software partitioning. In that case, the co-simulation stage is mainly used as verification of the chosen hardware/software partitioning and not as a design space exploration vehicle.

The Artemis simulation workbench applies a different, more flexible, approach which was initiated by Lieverse et al. [11] and, in a different context, Pimentel et al. [12]. Basically, our approach captures the two design stages discussed above (the early application evaluation stage and the system architecture evaluation stage) into one simulation methodology. This is illustrated in Figure 2(b). An *application model* describes the functional behavior of an application independently from architectural specifics,

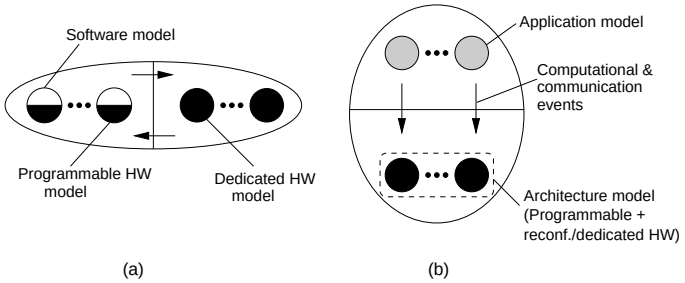

Fig. 2. (a) Typical hardware/software co-simulation versus (b) the Artemis approach.

assumptions on hardware/software partitioning or timing characteristics. Subsequently, the application model generates computational and communication events for the *architecture model*. This architecture model, which captures the timing behavior of the architecture, can simulate the performance consequences of the application events for both software execution (by a programmable component) or reconfigurable/dedicated hardware execution. So, unlike traditional co-simulation in which the software and hardware simulation are regarded as the co-operating parts, we explicitly distinguish *application simulation* and *architecture simulation* where the latter involves simulation of programmable, reconfigurable and dedicated hardware components. As a consequence, a single application model can be used to exercise different hardware/software partitionings or even to evaluate alternative underlying architectures. A more detailed description of our simulation methodology follows in the next section.

Another issue that most of today's co-simulation research efforts fail to address is how architecture models can be refined in a seamless manner across multiple levels of abstraction and how mixed-level architecture models (in which various architecture components are modeled at different levels of abstraction) are best supported. These are regarded as key issues in Artemis, which are discussed in detail in [13].

Regarding our reconfigurable architecture research, the majority of related work focuses on the following two subjects: 1) the design process from application to actual FPGA hardware and 2) run-time reconfiguration (RTR) of FPGAs. Research on the first subject includes shortening the design time and automating the design process [14]. While this subject remains to be an important one, most of the current research effort is put into run-time reconfiguration of FPGAs. The main motivation behind run-time reconfiguration is that hardware resources are still expensive and this can somewhat be compensated by re-using hardware. The research in this area focuses on decreasing the reconfiguration time [15] and architectural design of interconnection networks and FPGA modules [16], [17], [18].

Summarizing these efforts, it can be observed that they use fine-grain FPGA arrays which do not fully target multimedia applications and, moreover, target very specific algorithms. We intend to use coarser-grain FPGA structures which are better suited for the multimedia application domain and the multiplicity of algorithmic requirements associated with it.

IV. The simulation workbench

Based on previously gained experience, such as from the Spade [11] and Mermaid [12] projects, we strongly believe that the recognition of separate application and architecture models for the simulation of embedded systems architectures (as is illustrated in Figure 2b) plays a key role in establishing a high degree of modeling and exploration flexibility. Essential in this approach is that the application model is *architecture independent.* This implies that a single application model can be used for, or *mapped onto*, a range of architecture models, possibly representing different system architectures or simply modeling a single system architecture at various levels of abstraction.

After mapping, an application model can be co-simulated with an architecture model by means of trace-driven simulation. To this end, the application model generates traces of computational and communication events, describing application behavior, which are consumed by the trace-driven architecture simulator. The resulting co-simulation subsequently allows for evaluating the system performance for a particular application, input data-set, mapping and underlying architecture. In the remainder of this section, we will first describe how applications are modeled in Artemis after which the mapping and architecture modeling are explained.

A. Application modeling

For the modeling of applications, we use Kahn Process Networks [19]. To obtain a Kahn application model, a sequential application (written in C/C++) is restructured into a program consisting of parallel processes communicating with each other via unbounded FIFO channels. In the Kahn paradigm, reading from channels is done in a blocking manner, while writing is non-blocking. To generate the application events which drive the architecture simulator, the code of the Kahn application is instrumented with annotations. These annotations generate computational events, which may be as abstract as '*compute a DCT*', and communication events specifying from/to which (software) channels data is read or written.

The choice for Kahn Process Networks as application model is motivated by the fact that Kahn models nicely fit with the dataflow application domain (to which most of our target applications belong) and that they are deterministic. The latter means that the same application input always results in the same application output. So, the functionality of a Kahn application is not affected by architectural latencies, i.e. the application behavior is architecture independent. This is essential to guarantee the validity of the event traces when the application and architecture simulators are executed independently of each other [20].

As a starting point, we use the Yapi [21] framework for the implementation of the Kahn application models. Yapi provides the annotation functions which handle the Kahn interprocess communication and the generation of trace events. It also contains an application simulation engine which allows to concurrently execute the Kahn processes using threads. Currently, the abstraction level of annotations in Yapi directly relates to the level of generated application events. Since the abstraction level of the application events should match with the amount of detail in the architecture model, a refinement of the architecture model often implies that the annotations in the application model also need to be refined. Evidently, such correlation between the application and architecture models is undesirable for the Artemis simulation workbench, in which we strive for application models that transparently support architecture simulation at *all* of the required levels of abstraction.

We aim at improving the application modeling such that it is possible to generate application events at multiple levels of abstraction without the need to change the annotations in the application model. A possible solution which we are investigating is to introduce a separate *hierarchical annotation description* which describes how finer grained application events are generated from a particular coarse grained annotation. In this approach, the application model is instrumented with high-level annotations referring to coarse grained computations. Like in Yapi, these annotations can directly generate the application events for abstract architecture models. But, using the hierarchical annotation description, coarse grained events such as 'compute a DCT' can also be translated into several subtask events (e.g. one for each loop in the DCT computation) or even to instruction-level events which may include detailed information such as address references. The latter type of application events can, for instance, be used for the evaluation of cache and TLB performance of microprocessor components in a heterogeneous architecture. Two important characteristics of this hierarchical annotation scheme are the preservation of architecture independence of the annotated application model (i.e. it will not be polluted by low-level, architecture dependent, annotations) and extendibility. The latter implies that the modeler only needs to describe the annotation-hierarchy up to the level

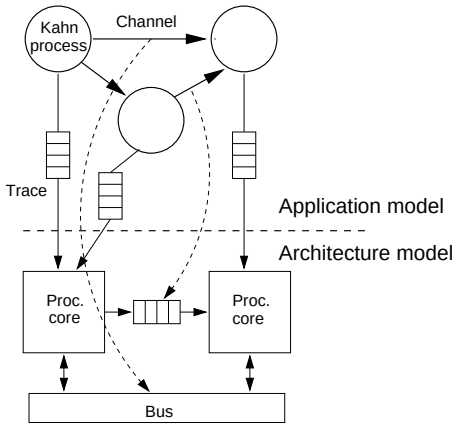

Fig. 3. Mapping a Kahn application model onto an architecture model.

of interest. So, if one is not interested in instruction-level events, then they can simply be left out in the annotation description.

B. Mapping and architecture modeling

Both the Kahn processes and channels of the application model are explicitly mapped onto the hardware components from the architecture model. With the term *mapped*, we mean that the generated trace of events from a specific Kahn process is routed towards a specific component inside the architecture model by using a trace-event queue. The Kahn process dispatches its application events to this trace queue while the designated component in the architecture model consumes them. This is illustrated in Figure 3. Mapping the FIFO channels between Kahn processes (shown by the dashed arrows in Figure 3) defines which communication medium at the architecture level is used for the data exchanges. In our example of Figure 3, one Kahn application channel is mapped onto a hardware FIFO buffer between the two processing components while another one is mapped onto the bus connection. One application channel stays unmapped since both its application tasks are mapped onto the same processing component.

As shown above, it is possible to map the event traces of two or more Kahn processes onto a single architecture component (e.g. when several application tasks are mapped onto a microprocessor). In that case, the incoming event traces need to be scheduled according to a user-supplied policy. Reversibly, Kahn processes should also be able to dispatch application events to multiple event queues such that a single application task can be mapped onto multiple architecture components (e.g. when refining an architecture model such that a component is decomposed into a number of more detailed components).

The underlying architecture model is solely a performance model and is therefore non-functional. This is possible because the functional behavior is already captured in the application model, which subsequently drives the architecture simulation. If the application behavior is data dependent, then the traces of application events also depend on the input data. The architecture model is constructed from generic building blocks provided by a library. This library contains performance models for processing cores, communication media (like busses) and different types of memory. At a high level of abstraction, the model of a processing core is a *black box* which can model a programmable processor, a reconfigurable component or a dedicated hardware unit. This is accomplished by the fact that the architecture simulator assigns the latencies to the incoming computational application events. To model software processing of an application event, a relatively high latency can be assigned to the event, as compared to when the application event would have been handled by dedicated or reconfigurable hardware. So, by simply varying the latencies for computational application events, different hardware/software partitionings can be evaluated.

In this approach, the communication application events (reads and writes to Kahn FIFO channels) are used for modeling the performance consequences of data transfers and synchronizations by writing to and reading from the appropriate communication medium at the architecture level. Unlike in the application model where all FIFO channels are unbounded, writes at the architecture level may also be blocking dependent on the availability of resources (e.g. buffer space).

As design decisions regarding hardware/software partitionings are made, the application events may be refined and may become increasingly geared towards a certain architecture implementation, like instruction-level events are oriented towards a microprocessor implementation. Likewise, the components of the architecture model may be refined such that they start reflecting the characteristics of a particular implementation (e.g. dedicated versus programmable hardware).

B.1 Spade and Sesame

For the actual implementation of the architecture models, the Artemis simulation workbench provides two simulation trajectories by means of the *Spade* (System-level Performance Analysis and Design space Exploration) [11] and *Sesame* (Simulation of Embedded System Architectures for Multi-level Exploration) [22] frameworks. As we will explain, the two frameworks are supplementary rather than competitive and have their own merits which justify their presence in the Artemis workbench.

Spade is the product of a research cooperation between

the Delft University of Technology, University of Leiden and Philips Research. It provides a small library which contains a black box model of a processing core, a generic bus model, a generic memory model and several interfaces for connecting these model building blocks with each other. All components are implemented using the cycle-based TSS simulation system [23], which is a Philips in-house product. The use of TSS exploits the fact that Philips has a large user community applying TSS for the implementation of cycle-true architecture simulators. Evidently, sharing a common simulation backbone significantly simplifies the transition from high-level Spade models to detailed TSS architecture models. For this reason, Spade envisions a trajectory which is shown at the left-hand side of Figure 4. The application model is mapped onto a high-level architecture model, which operates on relatively coarse grained application events (such as 'compute DCT'). This type of models allows for flexible exploration and analysis of issues such as hardware/software partitionings, communication bottlenecks and different communication structures. In the next step, when several design decisions have been made and a more accurate simulation of certain system components is required, a detailed simulator for these components is embedded into the overall architecture simulator. This is shown in the second stage of the Spade trajectory in Figure 4, where process A will be implemented in software on a microprocessor. Instead of mapping this process onto an abstract Spade model of a processing core, the latter is substituted for a detailed instruction-level simulator which emulates the actual code for process A. The process of embedding more detailed simulators is continued such that more and more functionality is gradually incorporated into the architecture model. At a later stage, this architecture simulator may then be used as a starting point for traditional (low-level) co-simulation as shown in Figure 2(a).

Spade's major strengths are its simplicity and flexibility (using only a few model building blocks, a large variety of system-level performance studies can rapidly be conducted) and its easy interfacing to more detailed TSS models. Its current weaknesses, which are addressed by the Sesame trajectory, are twofold. First, the step from an abstract Spade architecture model to a detailed (e.g. cycle-true) simulator of a component is rather large. Clearly, this step may require a substantial software engineering effort when such a detailed simulator is not yet available. Second, TSS's cycle-based nature unquestionably is effective for cycle-true simulations, but it may not always be the best simulation method for the black box architecture models which are used for the early design stages. In these abstract architecture models, the events which are important to the performance prediction usually occur with non-uniform time steps and may therefore be separated by large time frames. For example, a coarse grained application event such as 'compute DCT' typically requires the simulation of a multi-cycle latency at the architecture level. Consequently, cycle-based TSS will induce extra overhead for simulating every single cycle of such multi-cycle latencies.

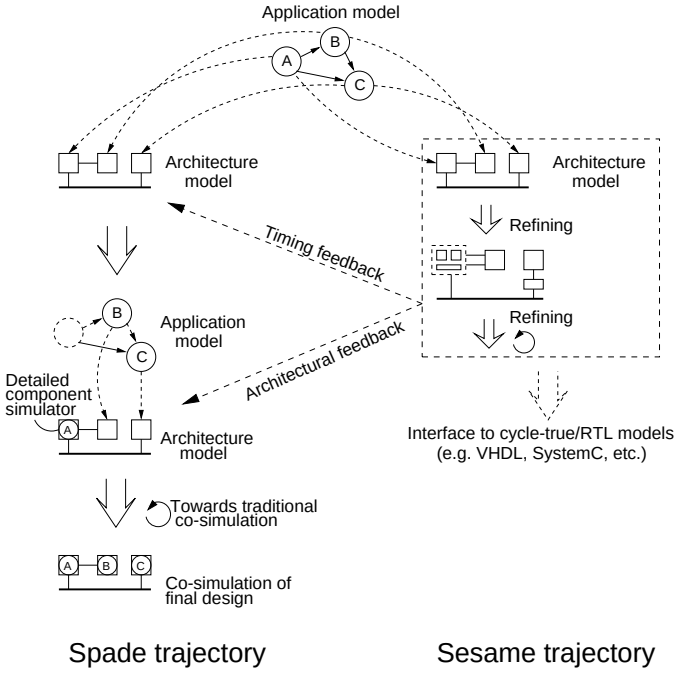

Fig. 4. The Artemis simulation workbench embodied by the Spade and Sesame frameworks.

The second simulation trajectory in Artemis, provided by the Sesame framework, is a research effort from the University of Amsterdam in close collaboration with the Spade team. Instead of using TSS, the Sesame architecture models are implemented in the discrete-event simulation language Pearl [24]. This is a relatively small but powerful object-based language which has specifically been designed with the purpose of (abstract) computer architecture modeling in mind. As a consequence, Pearl has shown to be extremely suitable for easily and quickly building new or extending existing architecture models [25], while also yielding simulation speeds of up to an order of magnitude faster than those obtained by more general-purpose simulation languages. At the downside, Pearl lacks the easy interfacing to more detailed simulators like Spade with its TSS models provides.

Discrete-event simulation, as performed by Pearl, is in contrast to cycle-based simulation well suited for abstract architecture models. It therefore overcomes the potential performance drawback of TSS and improves the scope of the design space that can be explored in a reasonable

amount of time. The Sesame trajectory, which is depicted at the right-hand side of Figure 4, exploits the high simulation speed of Pearl as well as the ease and swiftness with which it can construct and extend architecture models. Starting from the same black box architecture models as in Spade, Sesame allows for gradually refining these models. To this end, it provides an architecture model library which is more extensive than that of Spade as it includes models for architecture components at several levels of abstraction. This means that there will be, for example, multiple instances of a microprocessor model such as a black box model, a model which accounts for the performance consequences of the processor's memory hierarchy (e.g. TLB and caches) and a model which accounts for the performance impact of both its memory hierarchy and datapath (e.g. pipelining and ILP). Naturally, the more detailed models require the application model to be able to generate more detailed application events. The Sesame architecture models are less generic than those from Spade as they are not fixed building blocks with pre-defined interfaces but merely are template models which can be extended and adapted to the user's likings. This slightly increases the effort needed for building the architecture models, but it allows for more modeling flexibility which can be helpful when refining the models.

The above preconditions result in two research challenges which are addressed in Sesame. First, it should be possible to adjust the level of abstraction of the architecture models in a seamless and graceful manner. Refinement of one component in the architecture model should not lead to a totally renewed implementation of the entire model. Ideally, only the models of those architecture components which need to be refined are replaced in a 'plug-and-play' fashion by more detailed models. This embedding of refined, and possibly decomposed, component models in an otherwise unchanged architecture model seriously affects the interconnection with the rest of the architecture model. The second research challenge, which can be regarded as a direct consequence of the first one, involves the support for mixed-level simulation. This means that the abstraction level at which the different architecture components are modeled does not necessarily need to be the same. In [13], we show that transactions between architecture components form the main obstacle for mixed-level simulation.

As was shown, both Spade and Sesame contribute to the Artemis simulation workbench in their own way: Sesame purely focuses on quantitative performance evaluation whereas Spade is more open to the later stages of the design trajectory by applying the TSS simulation backbone. For Sesame, we are also considering some basic support (e.g. model wrappers) for interfacing it to more traditional cycle-true and RTL simulators such that the latter can import Pearl models. Also, it is imperative to have an interface between the Spade and Sesame trajectories which is well defined and for which support is provided to effectively utilize it. Figure 4 illustrates a possible interfacing scenario in which the Sesame trajectory can provide two types of feedback to the Spade trajectory. First, refined Sesame models may give timing feedback to the high-level Spade models in order to fine-tune the latencies they account for the different application events. Second, the design space exploration performed by Sesame allows for qualitative feedback on architectural specifics. This feedback may range from information that guides the construction and parameterization of low-level TSS models to (semi-)automatic generation of TSS-based equivalents of the Pearl models. The latter could be realized using an architecture model description language that is able to generate both TSS and Pearl architecture models.

To drive the realization and fine-tuning of the previously discussed application and architecture modeling methods, we initiated the *Startemis case study*. In Startemis, a modified M-JPEG encoder application, which includes dynamic quality control and can operate on both RGB and YUV formats on a per-frame basis, is modeled and studied for a number of target architectures. In [26], some initial results from the Startemis case study are presented.

V. The reconfigurable architecture framework

The hardware/software co-design paradigm is still often simplified to a strict binary decision, for either a dedicated function-specific 'co-processor', or a general purpose software programmable CPU. As also hinted in Figure 2, the Artemis project aims at supporting a sliding scale of options along the hardware/software axis. So next to pure and frozen function-specific dedicated hardware units, such units can be more flexible in their functionality by making these slightly programmable. On the other end, a general purpose programmable CPU can be tailored towards a specific application domain by adding domain-specific instructions and/or reconfigurable functional units. In terms of required electrical energy to execute a certain function (the power efficiency), both ends of the hardware/software scale can differ by two orders of magnitude. This is denoted in Figure 5. Having a choice for intermediate points in the hardware/software design trade-off, can help considerably for reaching an efficient architecture during design space exploration, and can influence the application partitioning decisions.

To obtain a higher efficiency for CPUs in the domain

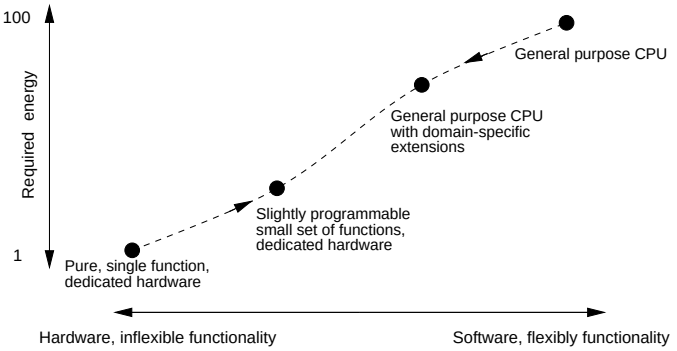

Fig. 5. Hardware/Software trade-offs and power consumption.

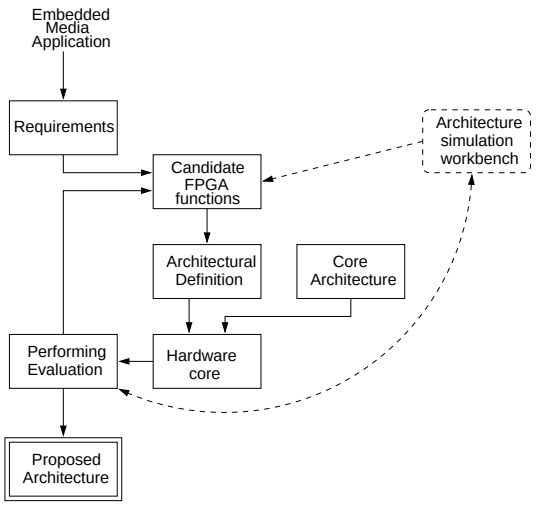

Fig. 6. The FPGA investigation framework.

of media processing, a conventional approach is to add specific 'media instructions' to the instruction set of the CPU. They provide increased performance due to the combination of 1) implementing small kernel functions which would otherwise require a couple of atomic operations, and 2) implementing a vector-style (single instruction multiple data) parallelism by executing this function in parallel on byte or half-word sections inside a full word. However, the size of the instruction set of a CPU (the number of different opcodes) is bounded by restrictions on fixed instruction formats and complexity of its decoder. Therefore the design of a media instruction set is a trade-off between generality and performance gain of specific instructions in the selected application domain.

A novel approach is to add reconfigurable hardware as a functional unit to the CPU (e.g. [27]). This allows to adapt the instruction set on the fly, individually optimized for each application. Potentially, this improves performance due to the virtually infinite choice of instructions. As others, we use the term FPGAs when referring to reconfigurable components. However, other innovative circuit techniques supporting reconfiguration may also be considered in Artemis. Such circuit techniques would affect important parameters like configuration element grain size, kernel function complexity, reprogramming speed, achievable clock frequency and power efficiency.

Integrating a reconfigurable component with a programmable core, like a VLIW processor, is the main target of our research. In particular the combination of reconfigurable components with a media-optimized VLIW core is challenging. For this purpose, the quantitative evaluation of the performance of a reconfigurable component needs to be addressed. In addition, the existing programming model needs to be extended in order to support the reconfiguration. The research effort will therefore result in proposals for new micro-architectures that include reconfigurable components together with the associated programming model. For this work, the proposed architecture simulation workbench will be used for architectural performance evaluation at the system level. For more detailed performance studies of the extended CPU, the workbench can be supplemented with a cycle-true TSS model of the new TriMedia VLIW core from Philips Research [28]. The resulting efficiency of the CPU combined with reconfigurable components, could influence the mapping of the application tasks onto architecture components.

More specifically, we will proceed as described in Figure 6 and we will investigate the following: First, we will identify and examine embedded media applications to determine if there are functions that occur in certain sequences that will provide some advantages when implemented in FPGAs. Examples for such functions are: IDCT, FFT, filters, huffman (de-)coders, etc. Consequently, we will determine the functional behavior of the FPGA structure needed to implement such functions and the primitive instructions required to be implemented in an FPGA logic design style together with the primitive program that is implemented in the FPGA. In combination with the available VLIW core and the FPGA programming of the functions, a hardware model for the processing unit will be determined. Consequently, the effectiveness of the reconfigurable hardware processing unit in the overall system architecture will be evaluated using the architecture simulation workbench. In case satisfactory results are obtained, new architecture extensions will be proposed to the design teams. In case the results are not satisfactory or there are more candidates to consider, additional effort will be dedicated to determine candidate functions or to update the configurable component circuit style. Some initial results of our reconfigurable architecture research are presented in [29].

VI. Conclusions

Because of the growing need for supporting multiple applications and standards, modern embedded systems architectures increasingly become heterogeneous, in which dedicated hardware components provide high performance for time critical tasks and programmable components provide flexibility. In this paper, we presented the Artemis project in which methods and techniques are developed to support the design of highly programmable heterogeneous embedded systems targeting the multimedia application domain. As a result, an architecture workbench is developed consisting of both a simulation environment for flexible architectural design space exploration and an experimentation framework for reconfigurable architectures.

The simulation environment allows for swift evaluation of different architecture designs, application-architecture mappings and hardware/software partitionings at various levels of abstraction and for a broad range of applications. This evaluation flexibility is obtained by recognizing separate application and architecture models in the system simulation. The reconfigurable architecture framework investigates the potentials of reconfigurable architectures as a new means to enhance the programmability of embedded systems without compromising performance. By combining the two frameworks, computationally intensive tasks can be identified in applications which are suitable for execution on a reconfigurable hardware component.

References

[1] W. H. Wolf, "Hardware-software co-design of embedded systems," *Proc. of the IEEE*, vol. 82, no. 7, pp. 967–989, July 1994.

[2] J. Rowson, "Hardware/software co-simulation," in *Proc. of the Design Automation Conference*, 1994, pp. 439–440.

[3] H. Hübert, "A survey of hw/sw cosimulation techniques and tools," M.S. thesis, Royal Inst. of Tech., Sweden, June 1998.

[4] S. L. Coumeri and D. E. Thomas, "A simulation environment for hardware-software codesign," in *Proceedings of the Int. Conference on Computer Design*, Oct. 1995, pp. 58–63.

[5] M. Bauer and W. Ecker, "Hardware/software co-simulation in a VHDL-based test bench approach," in *Proc. of the Design Automation Conference*, 1997.

[6] A. R. W. Todesco and T. H-Y. Meng, "Symphony: A simulation backplane for parallel mixed-mode co-simulation of vlsi systems," in *Proc. of the Design Automation Conference*, 1996.

[7] R. K. Gupta, Jr. C. N. Coelho, and G. De Micheli, "Synthesis and simulation of digital systems containing interacting hardware and software components," in *Proc. of the Design Automation Conference*, June 1992.

[8] C. Passerone, L. Lavagno, M. Chiodo, and A. Sangiovanni-Vincentelli, "Fast hardware/software co-simulation for virtual prototyping and trade-off analysis," in *Proc. of the Design Automation Conference*, 1997.

[9] K. Hines and G. Borriello, "Dynamic communication models in embedded system co-simulation," in *Proc. of the Design Automation Conference*, 1997.

[10] J-P Soininen, T. Huttunen, K. Tiensyrjä, and H. Heusala, "Cosimulation of real-time control systems," in *Proceedings of the EuroDAC 1995*, 1995, pp. 170–175.

[11] P. Lieverse, P. van der Wolf, E. F. Deprettere, and K. A. Vissers, "A methodology for architecture exploration of heterogeneous signal processing systems," in *Proc. of the Workshop on Signal Processing Systems*, 1999, pp. 181–190.

[12] A. D. Pimentel and L. O. Hertzberger, "An architecture workbench for multicomputers," in *Proc. of the 11th Int. Parallel Processing Symposium*, April 1997, pp. 94–99.

[13] A. W. van Halderen, A. Belloum, and A. D. Pimentel, "On hybrid abstraction-level models in architecture simulation," in *Proceedings of the Progress workshop*, Oct. 2000.

[14] I. Page, "Reconfigurable processor architectures," *Microprocessors and Microsystems*, vol. 20, no. 3, pp. 185–196, May 1996.

[15] A. DeHon, "Dynamically programmable gate arrays: A step toward increased computational density," in *Proc. of the Canadian Workshop on Field-Programmable Devices*, 1996, pp. 47–54.

[16] C. Alippi, W. Fornaciari, L. Pozzi, and M. G. Sami, "A DAG-based design approach for reconfigurable VLIW processor," in *Proc. of Design, Automation & Test*, March 1999, pp. 778–780.

[17] M. J. Wirthlin and B. L. Hutchings, "A dynamic instruction set computer," in *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines*, Apr. 1995, pp. 99–107.

[18] J. R. Hauser and J. Wawrzynek, "Garp: A MIPS processor with a reconfigurable coprocessor," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, Apr. 1997, pp. 24–33.

[19] G. Kahn, "The semantics of a simple lamguage for parallel programming," in *Proceedings of the IFIP Congress 74*, 1974.

[20] M. Dubois, F. A. Briggs, I. Patil, and M. Balakrishnan, "Trace-driven simulations of parallel and distributed algorithms in multiprocessors," in *Proc. of the 1986 Int. Conference in Parallel Processing*, Aug. 1986, pp. 909–915.

[21] E. A. de Kock, G. Essink, W. J. M. Smits, P. van der Wolf, J. Y. Brunel, W. M. Kruijtzer, P. Lieverse, and K. A. Vissers, "Yapi: Application modeling for signal processing systems," in *Proc. of the Design Automation Conference*, June 2000, pp. 402–405.

[22] A. D. Pimentel, A. W. van Halderen, L. O. Hertzberger, and P. van der Wolf, "Sesame: Simulation of Embedded System Architectures for Multi-level Exploration," Tech. Rep. (internal Artemis report), University of Amsterdam, 2000.

[23] W. Kruijtzer, "TSS: Tool for system simulation," *IST Newsletter, Philips Internal Publication*, vol. 17, pp. 5–7, Mar. 1997.

[24] H. L. Muller, *Simulating computer architectures*, Ph.D. thesis, Dept. of Comp. Sys, Univ. of Amsterdam, Feb. 1993.

[25] A. D. Pimentel, *A Computer Architecture Workbench*, Ph.D. thesis, Dept. of Computer Science, Univ. of Amsterdam, Dec. 1998.

[26] T. Stefanov, P. Lieverse, E. F. Deprettere, and P. van der Wolf, "Y-chart based system level performance analysis: an M-JPEG case study," in *Proceedings of the Progress workshop*, Oct. 2000.

[27] S. Wong, S. Cotofana, and S. Vassiliadis, "General-purpose processor huffman encoding extension," in *Proc. of the Int. Conference on Information Technology: Coding and Computing (ITCC 2000)*, March 2000, pp. 158–163.

[28] J. T. J. van Eijndhoven, F. W. Sijstermans, K. A. Vissers, E. J. D. Pol, M. J. A. Tromp, P. Struik, R. H. J. Bloks, P. van der Wolf, A. D. Pimentel, and H. P. E. Vranken, "Trimedia CPU64 architecture," in *Proc. of the IEEE Int. Conf. on Computer Design (ICCD '99)*, Oct. 1999, pp. 586–592.

[29] M. Sima, S. Vassiliadis, S. Cotofana, J. T. J. van Eijndhoven, and K. A. Vissers, "A taxonomy of custom computing machines," in *Proceedings of the Progress workshop*, Oct. 2000.