To appear in Proc. of the IEEE Workshop on Signal Processing Systems, Seoul, Korea, Aug. 2003.

UTILIZING SYNTHESIS METHODS IN ACCURATE SYSTEM-LEVEL EXPLORATION OF HETEROGENEOUS EMBEDDED SYSTEMS

Cagkan Erbas *Andy D. Pimentel*

Dept. of Computer Science, University of Amsterdam
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands
{cagkan,andy}@science.uva.nl

ABSTRACT

In this paper, we present a new mapping strategy for efficient design space exploration of heterogeneous embedded systems. The new mapping approach is developed within the context of the `Sesame` framework [1], [2] to improve its simulation accuracy and capabilities. In `Sesame`, separate application and architecture models together with an explicit mapping step are recognized within a system simulation. Originally, `Sesame` maps application models onto architecture models using trace-driven co-simulation of both models. However, this approach is limited by the fact that information is lost in the linear application traces that drive the architecture model. This may hamper accurate architecture simulation. The new mapping approach, which is based on Integer-controlled Dataflow actors, removes these limitations at the cost of a slight increase in complexity regarding the mapping. We compare our new approach with the current state-of-the-art mapping strategies under multiple criteria. Also with examples and a simple experiment, we illustrate how we integrate our new mapping strategy into `Sesame`'s mapping layer.

1. INTRODUCTION

Modern embedded systems, like those for media and signal processing, increasingly need to be multi-functional and must support multiple standards. This can be achieved with a high degree of *programmability*, which is provided by the microprocessor technology as well as reconfigurable hardware. However, real-time performance requirements and constraints on cost and power consumption still require significant parts of these systems to be implemented in dedicated hardware blocks. As a result, modern embedded systems evolve as *heterogeneous systems*; i.e. they consist of components from a range of fully programmable processing cores to dedicated blocks for time-critical application tasks. With the recent developments in silicon technology, such heterogeneous systems are usually integrated on a single chip, yielding multiprocessor systems-on-chip (SoCs).

For these modern embedded systems, it becomes more and more important to have good tools for system-level exploration, especially at an early design stage where the design space is very large. At this stage, the designer is more interested to see system-level trade-offs between different design alternatives, rather than an elaborate simulation valid only for a specific design. On the other hand, the available common practice tools are generally cycle-accurate simulators useful for synthesis rather than exploration. Increasingly these simulators become unsuited for early design space explorations as they are unable to cope with the increase in system complexity. At this point, the exploration tools should provide rapid evaluation of different design alternatives, keeping the designer from making high investments in terms of manpower and simulation time.

Unfortunately, design space exploration tools are not mature enough to satisfy all these requirements. They are typically either over-detailed or too superficial for system exploration. To fill this gap, using synthesis methods in system exploration is gaining popularity. There exist two distinct tracks to achieve this goal: top-down or bottom-up. In the top-down approach, a relatively inaccurate exploration tool is equipped with methodologies borrowed from a synthesis tool. But this should be done with special care as one should refrain from adding details not useful in terms of exploration. An entirely opposite path is taken in the bottom-up approach in which a synthesis tool is taken as a basis and it is gradually simplified to achieve a rapid simulator. However, in both tracks, the ultimate goal is to achieve a rapid reliable simulator that can quickly evaluate a large design space and provide figures about system trade-offs (utilization of components, data throughput, communication media contention, etc.) with high accuracy.

In the `Sesame` framework [1], [2], which we develop within the context of the `Artemis` project [3], we aim at building an architecture workbench which provides modeling and simulation methods and tools for the efficient and accurate design space exploration of heterogeneous embedded media systems. This workbench should allow for rapid performance evaluation of different architecture designs, ap-

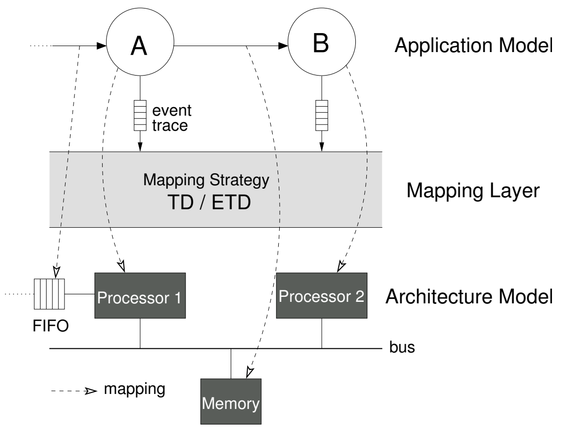

Fig. 1. The `Sesame` environment: modeling and exploring a simple producer/consumer communication is shown.

plication to architecture mappings, and hardware/software partitioning. In addition, it should do so at multiple levels of abstraction *and* for a wide range of multimedia applications. To achieve this flexibility, the `Sesame` framework recognizes separate application and architecture models together with an explicit mapping step to map an application model onto an architecture model. Currently, the `Sesame` environment, shown in Figure 1, uses the trace-driven (TD) approach for mapping in which the execution of an application model generates traces of application events that represent the application workload imposed on the architecture. However, besides being very flexible and efficient, this approach has a major drawback: the application information on control flow, parallelism, and data-dependency is lost. The loss of information may severely limit the accuracy and the capabilities of the simulation environment. In this paper, we present a new mapping strategy – which is based on Integer-controlled Dataflow actors [4] – to replace the TD approach in order to remove these drawbacks. With examples, we illustrate how we integrate our new mapping strategy into the `Sesame` environment.

The rest of this paper is organized as follows: next section discusses different approaches pursued for design space exploration, which is as well, a summary of the related work. Section 3 introduces our new mapping strategy and compares it with the current state-of-the-art mapping approaches under multiple criteria. In Section 4, we illustrate how we make use of our new mapping strategy with examples and the improvement on performance numbers with a simple experiment. Finally, Section 5 provides an overview of the current implementation and concludes the paper.

2. DESIGN SPACE EXPLORATION

Different approaches can be chosen for design space exploration, depending on the formalism. For example, Petri Nets and Process Networks are such formalisms frequently used in modeling of heterogeneous systems. CodeSign [5] is an environment where Petri Nets are combined with Object-Oriented concepts to build structural and composition mechanisms that are lacking in Petri Nets. However, Petri Nets are out of the scope of this paper.

Ptolemy [6] is an environment for simulation and prototyping of heterogeneous systems. It supports multiple models of computation (MoC) within a single system simulation. It does so by supporting domains to build sub-systems each conforming to a different MoC. Ptolemy supports an increasing set of MoCs, including all dataflow MoCs [7]: Synchronous Dataflow (SDF), Boolean Dataflow (BDF), Integer-controlled Dataflow (IDF), Dynamic Dataflow (DDF), together with the (Kahn) Process Network (KPN) MoC [8]. A number of these MoC's are also utilized in our `Sesame` environment: as we will explain, application behavior is modeled using the KPN MoC, the architecture-level operations are modeled using SDF actors, and the control flow and the data-dependent behavior within the Kahn processes are captured and conveyed to the architecture model level by IDF actors. Especially the utilization of SDF and IDF actors will become more clear in the given examples in Section 4.

A number of exploration environments such as VCC [9], Polis [10], and Milan [11] facilitate flexible system-level design space exploration by providing support for mapping a behavioral application specification to an architecture specification. However in `Sesame`, we push the separation of modeling application behavior and modeling of architectural constraints at the system-level to even greater extents. We do so by recognizing separate application and architecture models within a system simulation. An application model describes the functional behavior of an application, including both computation and communication behavior. The architecture model defines architecture resources and captures their performance constraints. An explicit mapping step maps an application model onto an architecture model for co-simulation.

Both the Spade [12] and Archer [13] environments show a lot of similarities with the `Sesame` environment in the sense that they share the same philosophy by recognizing separate application and architecture models. However, each of these environments uses its own architecture simulator and follows a different mapping strategy for co-simulation. In the next section, we discuss these mapping strategies and compare them under multiple criteria.

3. MAPPING STRATEGIES

Exploration environments making a distinction in application and architecture modeling need an explicit mapping to relate these models for co-simulation. In `Sesame`, we apply the trace-driven (TD) approach to carry out this task.

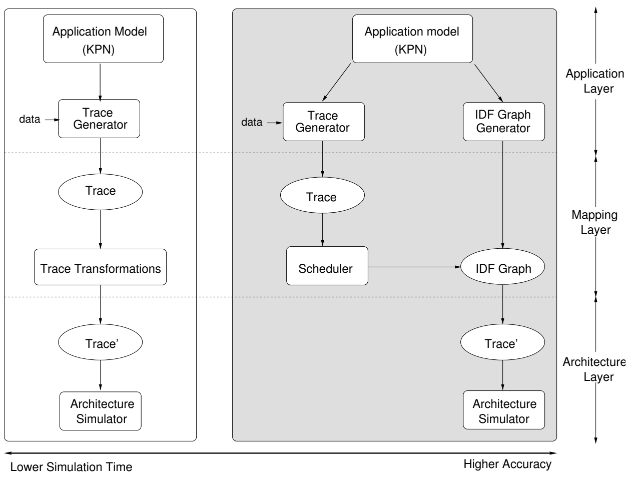

Fig. 2. The traditional Trace Driven approach versus our Enhanced Trace Driven approach.

In this approach, application processes implemented in the Kahn Process Network (KPN) MoC communicate with each other via FIFO channels. The workload of an application is captured by instrumenting the code of each Kahn process with annotations. By executing the application model with specific input data, each process generates its own trace of application events[1]. These events are coarse-grained operations like *read(pixel-block,channel-id)* and *execute(DCT)*. At the mapping layer, *application traces* are transformed into *architecture traces* which are subsequently used to drive architecture model components. Such a trace transformation guarantees a deadlock-free schedule at the architecture layer when application events from different Kahn processes are merged. The latter occurs when multiple Kahn application processes are mapped onto a single architecture model component. This mapping strategy is illustrated on the left side in Figure 2.

Exploration environments using the TD approach can fail to provide numbers with high accuracy. This is due to the loss of application information (about control flow, parallelism, dependency between events, etc.) which is present in the application code, but which is not present in the totally ordered application traces. The `Sesame` environment, being a spin-off from the Spade project [12], also inherits its TD approach. Recent research on `Sesame` [14], [15] showed us that refining architecture model components also requires refining the architecture events driving them. Again due to the information loss in the application traces from which the architecture events are derived, the latter is not always possible with the traditional TD approach. The only way to improve this is to change the mapping strategy. Since we start up with an exploration tool in hand, we follow the top-down approach. We take `Sesame` as a basis and enhance its mapping layer with a synthesis method, which in our case are IDF graphs [4]. The IDF MoC is powerful enough to perform system simulation, synthesis, and code generation. However, currently in `Sesame`, it is only used for simulation purposes. Further developments such as describing certain functions with IDF actors are within the possibilities of future research; we believe that this would open the way for a transition from design space exploration to system synthesis. With the current utilization of IDF actors for mapping, we move from the traditional TD approach to a new mapping strategy which we call the Enhanced Trace Driven (ETD) approach.

This new mapping strategy, illustrated on the right side in Figure 2, can be explained as follows: for each Kahn process at the application layer, we synthesize an IDF graph at the mapping layer. This results in a more abstract representation of the application code inside the Kahn processes. These IDF graphs consist of static SDF actors (due to the fact that SDF is a subset of IDF) embodying the architecture events which are the – possibly transformed – representation of application events at the architecture level. In addition, to capture control behavior of the Kahn processes, the IDF graphs also contain dynamic actors for conditional jumps and repetitions. IDF actors have an execution mechanism called *firing rules* which specify when an actor can fire. This makes IDF graphs executable. However, in IDF graphs, scheduling information of IDF actors is not incorporated into the graph definition, and it should be supplied via a scheduler explicitly. The scheduler in `Sesame` operates on the original application event traces in order to schedule our IDF actors. The scheduling can be done either in a static or dynamic manner. In dynamic scheduling, the application and architecture models are co-simulated using a UNIX IPC-based interface to communicate events from the application model to the scheduler. As a consequence, the scheduler only operates on a window of application events which implies that the IDF graphs cannot be analyzed at compile-time. This means that, for example, it is not possible to decide at compile-time whether an IDF graph will complete its execution in finite-time; or whether the execution can be performed with bounded memory[2].

Alternatively, we can also schedule the IDF actors in a semi-static manner. To do so, the application model should first generate the entire application traces and store them into trace files (if their size permits this) prior to the architectural simulation. This static scheduling mechanism is a well-known technique in Ptolemy and has been proven to be

[1]We will use "application events" and "application-level operations" interchangeably throughout the text. The same is true for the terms "architecture events" and "architecture-level operations".

[2]Many of the analysis problems in IDF converge to the halting problem of Turing machines. This is due to the power of the MoC. In [16], Buck shows that it is possible to construct a universal Turing machine with only using BDF actors (IDF is an extension of BDF, see [4]) together with actors for performing addition, subtraction and comparison of integers.

very useful for system simulation [4]. However in `Sesame`, it does not yield to a fully static scheduling. This is because of the fact that the SDF actors in our IDF graphs are tightly coupled with the architecture model components. An SDF actor sends a token to the architecture layer to initiate the simulation of an event. It is then blocked until it receives an acknowledgement token from the architecture layer indicating that the performance consequences of the event have been simulated within the architecture model. To give an example, an SDF actor that embodies a *write* event will block after firing until the *write* has been simulated at the architecture level. This *token exchange mechanism* yields a dynamic behavior. For this reason, the scheduling in `Sesame` is semi-static rather than static.

In the Archer project [13], a bottom-up approach is followed. A synthesis tool called Control Data Flow Graphs (CDFG) [17] is taken as a basis. However, the CDFG notation is too complex for exploration, so they move to a higher abstraction level called Symbolic Programs (SP). SPs are CDFG-like representations of Kahn processes. They contain control constructs like CDFGs, however unlike CDFGs, they are not directly executable. They need extra information for execution which is supplied in terms of *control traces*. These control traces (which are somewhat similar to the traces we use) are generated by running the application with a particular set of data. At the architecture layer, SPs are executed with the control traces to generate architecture traces which are subsequently used to drive the resources in the architecture model.

Table 1. A comparison of different mapping strategies.

Comparison criteria	*TD*	*ETD*	*SP*
Executable graphs	no	yes	no
Contain control flow information	no	yes	yes
Capture dependencies between events	no	yes	yes
Allow complex event transformations	no	yes	yes
Complexity of architecture models	low	low	moderate
Simulation time	low	moderate	moderate
Accuracy	low	high	high

In Table 1, we give a comparison chart for the three mapping strategies: TD, SP, and Enhanced Trace Driven (ETD). From this table, the advantages of ETD over TD should be clear as the former retains the application information that has been lost in the TD approach. This means that at the cost of a small increase of complexity at the mapping layer (and possibly a slight increase of simulation time) we now have the capability of performing accurate simulations as well as performing complex transformations (e.g. refinement) on application events. Comparing ETD to the SP approach, we see that they both allow for accurate simulations. In the ETD approach, all the complexity is handled at the mapping layer while in the SP approach it is spread over the mapping layer and the architecture model layer.

```
while(1) {
  read(in_NumOfBlocks,NumOfBlocks);

  // code omitted

  write(out_TablesInfo,LumTablesInfo);
  write(out_TablesInfo,ChrTablesInfo);
  switch(TablesChangeFlag) {
    case HuffTablesChanged:
      write(out_HuffTables,LumHuffTables);
      write(out_HuffTables,ChrHuffTables);
      write(out_Command1,OldTables);
      write(out_Command2,NewTables);
      break;
    case QandHuffTablesChanged:

      // code omitted

    default:
      write(out_Command1,OldTables);
      write(out_Command2,OldTables);
      break;
  }

  // code omitted

  for(int i=1;i<(NumOfBlocks/2);i++) {

    // code omitted

    read(in_Statistics,Statistics);
    execute("op_AccStatistics");

    // code omitted
  }
}
```

Fig. 3. Annotated C++ code for the Q-Control process.

The control traces and the SPs are created and manipulated (e.g. transformed) at the mapping layer, while they are also directed to the architecture layer to generate the architecture traces that derive the architectural simulation. This mechanism results in having more complex architecture model components in the SP approach which may hamper the architectural exploration. We also observe another advantage of the ETD approach over the SP approach that the graphs in the former are inherently executable while in the latter are not. This facilitates relatively easy construction of the system simulation.

4. EXAMPLES & A SIMPLE EXPERIMENT

In this section, we illustrate how we build IDF graphs with an example taken from an M-JPEG application we studied in [1]. In Figure 3, we present the annotated C++ code for the *Q-Control* Kahn process at the application layer. The Q-Control process computes the tables for Huffman encoding and those required for quantization for each frame in the video stream, according to the information gathered about the previous video frame. This operation of the Q-Control process introduces *data-dependent* behavior into the M-JPEG application. In Figure 4, a corresponding IDF graph is given for realizing a high-level simulation. That is, the architecture-level operations embodied by the SDF actors (shown with circles) represent *read*, *execute* and *write* operations (shown as *R*, *E*, and *W* in Figure 4, respectively). These SDF actors drive the architecture layer components by the token exchange mechanism (explained in Section 3); however, for the sake of simplicity, the architecture layer and the token exchange mechanism are not shown in the figure. The IDF actors CASE-BEGIN, CASE-END, REPEAT-BEGIN, and REPEAT-END model jump and repetition structures present in the application code. The scheduler reads an application trace, generated earlier by running the application code with a particular set of data, and executes the IDF graph by scheduling the IDF actors accordingly by sending the appropriate control tokens. In Figure 4, there are horizontal arcs shown with dotted lines between the SDF actors.

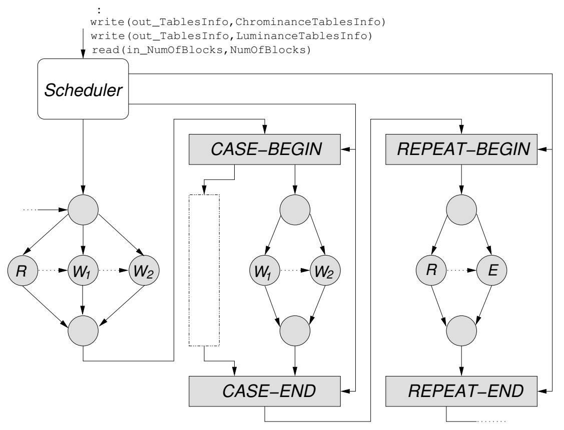
:
write(out_TablesInfo,ChrominanceTablesInfo)
write(out_TablesInfo,LuminanceTablesInfo)
read(in_NumOfBlocks,NumOfBlocks)
Scheduler
CASE–BEGIN
REPEAT–BEGIN
R
W1
W2
W1
W2
R
E
CASE–END
REPEAT–END

Fig. 4. IDF graph representing the Q-Control process from Figure 3 that realizes high-level simulation at the architecture layer.

Adding these arcs to the graph results in a sequential execution of architecture-level operations while removing them exploits parallelism. This is a good example that shows the flexibility of our mapping strategy.

In Figure 5, we present an IDF graph that implements *communication refinement* [14] in which communication operations, *read* and *write*, are refined in such a way that the synchronization parts become explicit. As a consequence, an application-level *read* operation is decomposed into three architecture-level operations: *check-data*, *load-data*, *signal-room* (shown as *cd*, *ld*, and *sr* in Figure 5, respectively), and similarly, an application-level *write* operation is decomposed into three architecture-level operations: *check-room*, *store-data*, *signal-data* (shown as *cr*, *st*, and *sd* in Figure 5, respectively). The computational *execute* operations are not refined, they are simply forwarded to the architecture model layer. This type of refinement not only reveals the system-bottlenecks due to synchronizations, but also makes it possible to correct them by reordering the refined operations (e.g., early checking for data/room or merging multiple costly synchronizations). In this case, SDF actors represent these refined architecture-level operations. So, by simply replacing the SDF actors with the refined ones in a plug-and-play fashion, one can realize communication refinement (and possibly other transformations). Once again, the level of parallelism between the architecture-level operations can be adjusted by adding or removing arcs between the SDF actors.

We have also performed a simple experiment in which we explored how far the performance numbers of the TD approach diverge from the numbers of the ETD approach. The setup for the experiment is already given by Figure 1. In this simple application, process A reads a block of data, performs a computation on it, and writes a block of data for process B. Simultaneously, process B reads a block of data and performs a computation on it. In this experiment, process A and process B are mapped onto Processor 1 and Processor 2, respectively. Processor 1 reads its input data from the dedicated FIFO while the communication between the two processors is performed through the shared memory. We consider the following scenario: we assume that Processor 1 has no local memory, thus before fetching data from the FIFO it should first consult the memory whether there is room for writing the results of the computation. This behavior requires communication refinement in the sense shown in Figure 5 which enables Processor 1 to perform early *check-room*. We modeled this behavior using the ETD approach and compared our performance results with the numbers obtained by the TD approach.

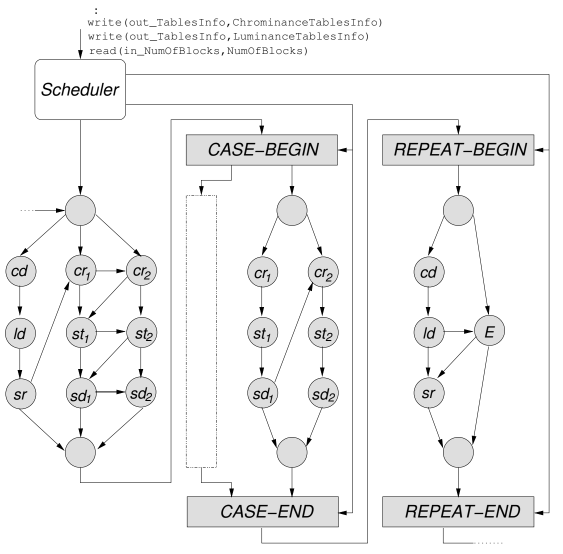
:
write(out_TablesInfo,ChrominanceTablesInfo)
write(out_TablesInfo,LuminanceTablesInfo)
read(in_NumOfBlocks,NumOfBlocks)
Scheduler
CASE–BEGIN
REPEAT–BEGIN
cd
cr1
cr2
ld
st1
st2
sr
sd1
sd2
cr1
cr2
st1
st2
sd1
sd2
cd
ld
E
sr
CASE–END
REPEAT–END

Fig. 5. IDF graph for the process in Figure 3 implementing communication refinement.

In Table 2, we present how much the numbers of the TD approach deviate from the numbers of the ETD approach which models the correct behavior. We should note that we have performed 16 simulation runs using four different speeds for the processors, s being the slowest and $4s$ being the fastest. In Table 2, these processor speeds are abbreviated as S_{P_1} and S_{P_2} for Processor 1 and Processor 2, respectively. From the simulation results we observe that the performance numbers obtained by the TD approach deviate between 18.7% and 44.6% from the correct numbers.

Table 2. Experimental Results

Error Ratio Matrix for TD mapping approach				
S_{P_1}xS_{P_2}	s	$2s$	$3s$	$4s$
s	44.6%	38.0%	29.7%	18.7%
$2s$	40.4%	43.2%	34.3%	22.2%
$3s$	32.4%	37.4%	40.7%	27.2%
$4s$	21.8%	25.9%	31.8%	35.2%

5. CONCLUSION & DISCUSSION

In this paper, we presented a new mapping strategy for efficient and accurate design space exploration of heterogeneous systems. To accomplish this, we followed a top-down approach in which we took a relatively low accurate exploration tool, in our case `Sesame`, and improved its mapping strategy with methodologies borrowed from a synthesis tool. The synthesis method, in our case the IDF MoC, has been proven to be very useful for exploration as well. We showed this with examples, where in each case, easy construction of IDF graphs was possible. With these examples, we showed how we can change the amount of possible parallelism by simply adding or removing a few arcs, while we also demonstrated how communication refinement can be realized by only replacing the SDF actors. Performing a simple experiment, we could also show how the new mapping approach improves the simulation accuracy.

Currently, we are experimenting with our new mapping strategy on real-life media applications. To do so, we have implemented the IDF and SDF actors as generic building blocks in Pearl [2], which is a small but powerful object-based simulation language that is also used for building architecture models in `Sesame`. The fact that both our IDF graphs and the architecture models are implemented in Pearl simplifies their interaction (i.e., token exchange mechanism). Concerning only the SDF actors, we have already performed a simple case study in which we modeled and analyzed different communication behaviors at the architecture level. We did this by simply changing the arcs between the SDF actors while leaving the application model unchanged [14].

6. ACKNOWLEDGEMENTS

This research is performed within the `Artemis` project funded by PROGRESS. The authors thank Ed Deprettere, Pieter van der Wolf and Simon Polstra for good discussions.

7. REFERENCES

[1] A. D. Pimentel et al., "Towards efficient design space exploration of heterogeneous embedded media systems," in *Embedded Processor Design Challenges: Systems, Architectures, Modeling, and Simulation*. Springer, 2002.

[2] J. E. Coffland and A. D. Pimentel, "A software framework for efficient system-level performance evaluation of embedded systems," in *Proc. of the ACM Symposium on Applied Computing*, March 2003.

[3] A. D. Pimentel et al., "Exploring embedded-systems architectures with Artemis," *IEEE Computer*, Nov. 2001.

[4] J. T. Buck, "Static scheduling and code generation from dynamic dataflow graphs with integer valued control streams," in *Proc. of the 28th Asilomar conference on Signals, Systems, and Computers*, Oct. 1994.

[5] R. Esser et al., "Using an object-oriented petri net tool for heterogeneous system design," in *Proc. of the Algorithmen und Werkzeuge für Petrinetze*, 1997.

[6] J. T. Buck et al., "Ptolemy: A framework for simulating and prototyping heterogeneous systems," *Int. Journal of Computer Simulation*, Apr. 1994.

[7] E. A. Lee and T. M. Parks, "Dataflow process networks," in *Proc. of the IEEE*, May 1995.

[8] G. Kahn, "The semantics of a simple language for parallel programming," in *Proc. of the IFIP Congress 74*, 1974.

[9] "Cadence Design Systems, http://www.cadence.com".

[10] F. Balarin et al., *Hardware-Software Co-design of Embedded Systems – The POLIS approach*, Kluwer Academic Publishers, 1997.

[11] S. Mohanty et al., "Rapid system-level performance evaluation and optimization for application mapping onto SoC architectures," in *Proc. of the IEEE ASIC/SOC Conference*, Sep. 2002.

[12] P. Lieverse et al., "A methodology for architecture exploration of heterogeneous signal processing systems," in *Proc. of the IEEE Workshop on Signal Processing Systems*, Oct. 1999.

[13] V. Živković et al., "Design space exploration of streaming multiprocessor architectures," in *Proc. of the IEEE Workshop on Signal Processing Systems*, Oct. 2002.

[14] A. D. Pimentel and C. Erbas, "An IDF-based trace transformation method for communication refinement," in *Proc. of the ACM/IEEE Design Automation Conference*, June 2003.

[15] A. D. Pimentel et al., "On the modeling of intra-task parallelism in task-level parallel embedded systems," To be published in Domain-Specific Processors: Systems, Architectures, Modeling, and Simulation, Marcel Dekker, 2003.

[16] J. T. Buck, *Scheduling Dynamic Dataflow Graphs with Bounded Memory using the Token Flow Model*, Ph.D. thesis, Dept. of EECS, UC Berkeley, 1993.

[17] W. Wolf, *Computers as Components – Principles of Embedded Computing System Design*, Morgan Kaufmann Publishers, 2001.