CoFrame: A Modular Co-Design Framework for Heterogeneous Distributed Systems

TIK-Report No. 81

Michael Eisenring, Eckart Zitzler, Lothar Thiele *

Abstract

This paper presents CoFrame, a novel modular co-design framework for heterogeneous distributed systems. The center of the approach is a pool of dynamic data structures which is analyzed, modified and refined by algorithms and tools to map a system specification to a feasible implementation consisting of arbitrary linked processors, DSPs and FPGAs. Due to its modular structure, CoFrame can be easily adapted to various control and data flow oriented problems by exchanging the domain-specific components. As an example we present a CoFrame tool configuration providing i) design space exploration for architecture synthesis using a state-of-the-art evolutionary algorithm for multiobjective optimization and ii) communication synthesis using an object-oriented approach. Finally, a case study of a molecular dynamic solver is described.

1 Introduction

As the design and implementation of heterogeneous embedded systems is growing in its complexity, there is a strong need for flexible and modular tool support. The system designer should focus on the functional behavior and correctness of the specification and got assistance in evaluating and implementing various solutions. This assistance especially includes *architecture synthesis* (allocation, binding and scheduling) and *communication synthesis*.

As experience shows, interfaces between heterogeneous components of a system are crucial for the overall system quality in terms of performance, cost and power consumption. On the other hand, the consideration of interface synthesis in an overall design flow causes problems as (1) the estimation of their properties depends on the particular implementation (FPGA family, processor family, processor type, bus or network protocol), (2) major constraints on binding and allocation are implied as not all combinations of interface implementations (e.g., memory mapped or interrupt driven) are possible and (3) storing interface implementations for all possible combinations of computation and communication links is not feasible.

*Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH), Zurich, email: {eisenring, zitzler, thiele}@tik.ee.ethz.ch

Moreover, we are faced with an increasing heterogeneity concerning the specification (e.g., programming languages, block diagrams, different models of computation such as synchronous data flow graphs, state machines, process networks), implementation (e.g., microcontroller, microprocessor, FPGA, DSP, different forms of communications such as busses or networks), scheduling methods (e.g., EDF, rate monotonic, static, quasi-static) and optimization criteria (e.g., power consumption, price, weight). All these factors depend on the particular application domain. These observations lead to some (additional) requirements for system synthesis methodologies. (1) It should be possible to adapt the design flow and the used tools to the particular application domain, e.g., estimation and scheduling algorithms and algorithm/architecture specification methods, and (2) previous knowledge about the system which is being designed must be included in the synthesis, e.g., restrictions on the communication structure, usable components and scheduling procedures. Knowledge about design flow, tools and design constraints is an important source of reuse. The meta-user which configures the design methodology for an application domain is part of the design flow and his activities must be supported.

In either areas a lot of work has been published. In [1] the problem of automatically determining an architecture composed of hardware library components (only connectable via simple ports) using genetic algorithms, simulated annealing and tabu search considering real-time constraints is studied. In [3] and [20] a system-level synthesis approach using an evolutionary algorithm is described. Its problem and architecture graph descriptions consider communication time and bus conflicts. [6] extends [3] and includes power consumption as an optimization goal. HiPART [14] implements a set of communicating C and VHDL processes onto a heterogeneous target platform [13] consisting of processor, ASIC, DSP and FPGA modules by using an interactive hierarchical partitioning algorithm. However, the approaches often omit an overall view of the system and neglect power consumption [1] [23], communication time [1], FPGAs [3] [6] [20] [23], interface selection [3] [20] or real design space exploration [14] [23].

Works concerning communication synthesis have been published by, e.g., [15] where various interfacing techniques have

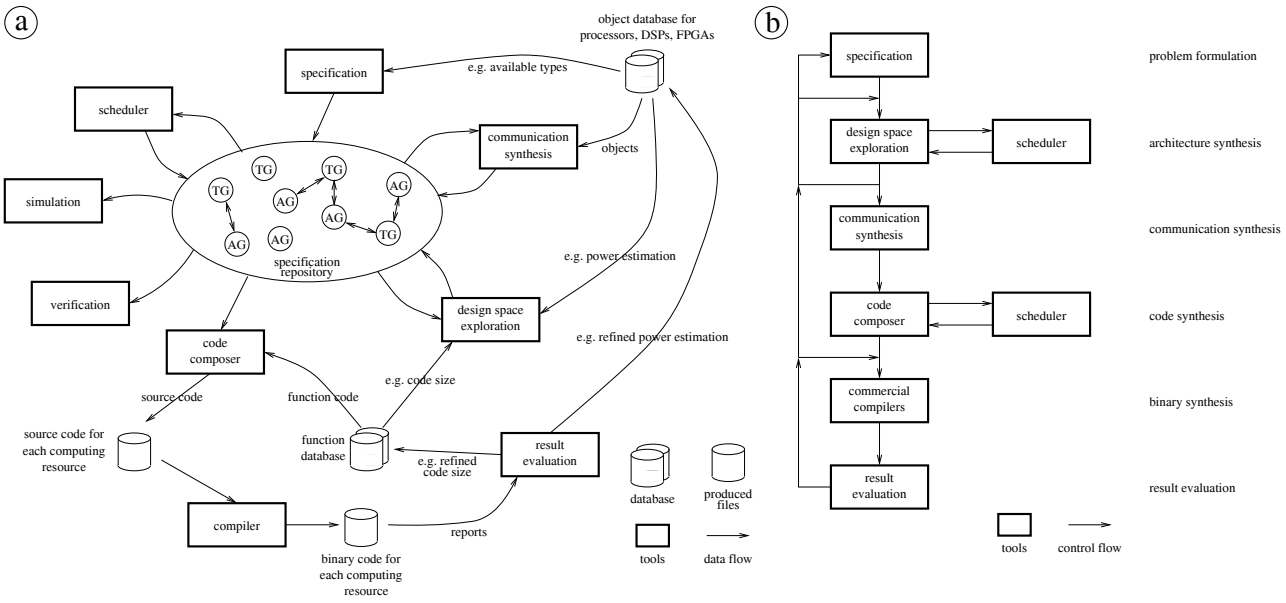

Figure 1. a) Data flow view b) Example of a tool configuration

been pointed out connecting re-usable IP (intellectual property) components. [17] focuses on arbitrary connected processors using common bus protocols and generates the communication between the processors. The result is an application-specific real-time OS for each processor but the approach neglects FPGAs as target. [22] proposes an object-oriented communication library of pre-implemented communication channels.

Frameworks for embedded system design include, e.g., Polis [2] and Ptolemy [4]. The CoWare approach [16] which focus on system-on-a-chip implementations and uses processes calling remote procedure calls for communication as behavior description. In Cosyma [9] the behavior is written in C^x a superset of C and is automatically partitioned between hardware and software by using a simulated annealing algorithm. Chinook [5] [18] maps a behavioral description of communicating processes on a single or multiprocessor system and generates software drivers and glue logic to connect processors and external chips but not for FPGAs. The SIERA co-design framework [19] maps a network of concurrent processes onto a printed circuit board given an architecture template comprising FPGAs, ASICs and processors. The above approaches either focus on specific problem domains, target architectures, provide no design exploration or do not include the concept of IP blocks.

In contrast to these frameworks that implement a domain specific design flow CoFrame provides a modular and flexible co-design framework with a configurable tool suite allowing an easy adaptation to control and dataflow oriented problems. Moreover, automatic interface and communication synthesis is integrated.

The main contributions of this paper are:

- Presenting the modular and flexible framework structure of CoFrame.
- Architecture synthesis is performed by an evolutionary algorithm (EA) that i) carries out multiobjective optimization in contrast to other studies [14][23], ii) uses a state-of-the-art multicriteria optimization technique as opposed to other design space exploration approaches (e.g., [6]), and iii) evolves interconnections and protocols as basis for the communication synthesis.
- Connecting the evolutionary algorithm to an object-oriented interface synthesis tool able to cope with standard interfaces as well as proprietary ones.
- Applying the framework to a molecular dynamic problem showing the continuous design flow from specification to implementation.

2 Methodology

The methodology is characterized by considering a typical meta-user which defines the domain specific design methodology. Two aspects are described in more detail here, namely the modular structure of the data flow between the tools and the customization of input specification and design flow.

The structure of CoFrame can be considered from two different views, namely a

- **data flow view** describing the data flow of *dynamic data* (modified during design process, e.g., different refinement levels of the specification) and *static data* (stable during design process, e.g., object-oriented models of processors) between the tools and a

- **tool configuration view** describing the tools orchestration to get a feasible solution for a given problem description.

The **data flow view** is depicted in Fig. 1a). In the middle is the specification repository which comprises:

- *Task graphs TG* of communicating tasks describing the behavior.
- *Architecture graphs AG* expressing the user's existing knowledge about a set of possible architectures (structural target description) to constrain the design space.
- *Mapping functions M* relating pairs of TG and AG allowing a set of architecture graphs to be used by several different behavior descriptions and vice versa.

The separation of behavior and architecture description, similar to [19], eases the design space exploration. On this repository the following distinct tools work.

Specification tools allow the user to specify task and architecture graphs including their relations. These representations may as well be extracted form other input forms such as programs or block diagrams. In addition, constraints which model the users knowledge about the application domain can be specified. For example, the feasible set of architectures can be coded into the architecture graphs, the admissible bindings into the mapping functions. This possibility generates additional constraints for the exploration phase and makes the optimization much more complex. Therefore, simple algorithms such as in [6] can not be used.

Tools for *design space exploration* (i.e., architecture synthesis) seek for optimal solutions of a given behavior considering various user constraints like task deadlines, power constraints, implementation cost etc. The tools have access to different databases providing estimation data about task functions and computing resources (processors, DSPs and FPGAs). In addition, the design space exploration includes estimation tools which estimate the different objective functions based on the current set of allocations and bindings. This exploration may also use *scheduler* tools for task and communication scheduling, see Section 3.

Communication synthesis tools build the necessary communication infrastructure on the target architecture according to the task execution model and generate hardware interfaces and device drivers based on the object database of computing resources, see Section 4.

The *code composer* tools allow to gather the information for each computing resource and produce appropriate source code comprising the generated device drivers and tasks described within a function library. These sources are compiled into hardware or software binaries by *compiler* tools. The *result evaluation* helps to rate the gained codes and provide the information to refine the estimations within the databases.

Tools for *simulation* and *verification* support the design process to validate and verify the specifications.

The **tool configuration view** specifies the design flow to implement and is dedicated to the users current problem domain. Figure 1b) shows an example of a tool configuration which has been implemented in CoFrame and is presented throughout this paper. As shown in Fig. 1b), several loops in the control flow of the tool configuration allow an iterative refinement of the system under consideration.

In order to facilitate the definition of a new design flow or problem specification, the CoFrame environment makes use of a new customizable graphical user interface and specification repository, see [12] (Modeling, Simulation and Evaluation of Systems), see snapshot in Fig. 2. Using a graph specification language, new graphical formalisms can be defined and the corresponding editors are automatically generated. The tool is used in CoFrame to specify the design flow, the algorithm and architecture specification and the mapping functions. In particular, the user's interfaces for the specification of task and architecture graphs and the specification repository are shown on the left half of Fig. 2. The right half of Fig. 2 shows the *Synthesis Manager* providing the tool configuration.

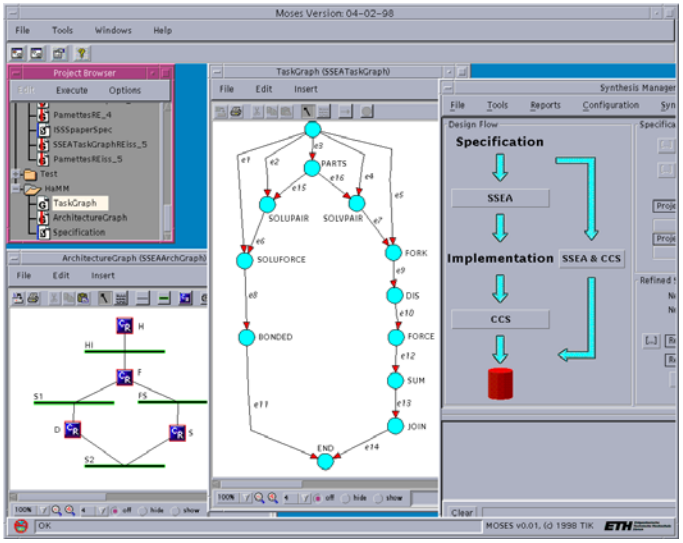

Figure 2. Snapshot of CoFrame

3 Design Space Exploration

Design space exploration aims at finding the *Pareto-optimal* set among all possible designs of a complex hardware/software system. This set of designs represents the optimal, alternative trade-offs between the often incommensurable and conflicting design criteria such as cost, latency, and power consumption. It is characterized by the fact that none of its members can be improved in one objective without degradation in another. On the basis of the Pareto-optimal front, the engineer can choose a final design which best fits the market requirements.

In our approach, the design space exploration tool consists of three components as depicted in Fig. 3 (the arcs represent the control flow between the distinct modules).

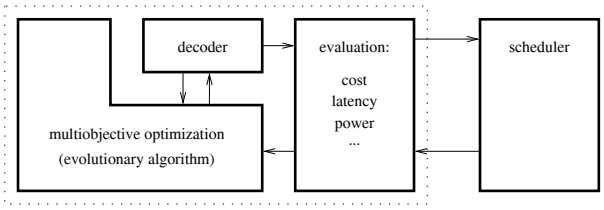

Figure 3. Design space exploration tool: internal structure

- The problem-independent multiobjective optimizer performs the exploration using an evolutionary algorithm (EA) and samples the space of possible allocations and bindings for Pareto-optimal solutions with regard to the given task graph and architecture graph as well as the design criteria.
- The decoder maps the encoded allocation and binding to a feasible design. Several problem-dependent constraints, e.g., the maximum area of a FPGA or the maximum number of partners connected to a particular bus, have to be taken into account. In order to reduce the number of infeasible solutions, it might be necessary to incorporate a repair heuristic based on domain knowledge that transforms infeasible designs to feasible ones and to use penalty functions in order to guide the search to feasible solutions.
- The estimator tries to evaluate the multiobjective fitness functions of the individual solutions within a population. To this end it uses not only the actual allocation and binding but also the corresponding implementation and function objects from the corresponding databases, see Fig. 1a. Examples of estimation tools which can be plugged in are scheduling algorithms, power and run-time estimators based on simulation and profiling of software on the allocated and bound target, or area and cycle time estimators in case of hardware targets. On this basis, the values for the different objectives are calculated and further constraints such as the violation of deadlines, pin or area constraints are checked. Finally, the evaluation result is transferred to the EA which ranks the current solutions using this information.

In the example of this paper, the design exploration phase is embedded twice into the design flow. At first, the allocation and binding of computation and communication resources is done such as busses, networks, microprocessors, DSPs and FPGAs. Based on this coarse grained information, the interfaces between these heterogeneous components are chosen, e.g., using direct I/O, interrupts or memory mapping. This refinement strategy reduces the size of the design space considerably. Again, this is an example where the need for a flexible tool structure and an open design flow is apparent.

The components of CoFrame can be exchanged independently. Even the EA might be replaced by another optimization method. However, up to now there are few if any alternatives to EA-based multiobjective optimization [11]. Moreover, EAs seem to be particularly suited for this task because they are capable of capturing multiple Pareto-optimal solutions in a single simulation run and might exploit similarities of solutions by recombination. Here, the Strength Pareto Evolutionary Algorithm (SPEA), a recent multiobjective EA [25][26], is used. It was shown empirically that SPEA outperforms other evolutionary approaches to multicriteria optimization on different sorts of application [21][24][26].

4 Communication Synthesis

The goal of the communication synthesis is to provide the communication infrastructure on the target architecture taking into account computing resources, communication modules and protocols selected by the EA during architecture synthesis. It establishes the connections between the heterogeneous computing resources by generating interfaces and device drivers. The information about communication modules and protocols is described by attributes assigned to nodes and edges of the task and architecture graph.

As pointed out in section 1 the connection between heterogeneous system components is crucial for the overall design flow and therefore automatic interface generation is necessary. In this tool configuration (see Fig. 1b) CoFrame makes use of HASIS [7] [8] an object-oriented toolset for automatic hardware software interface synthesis. Each computing resource of the architecture graph has an attribute *TYPE* referring to an object of the object database. Each of these objects models a real chip and possesses a core object representing the computing engine and one or more IO objects [7] modeling the IO-signaling facilities, e.g., bus interface, serial link, dedicated port, etc. The main feature of the IO objects are built-in code generators able to generate various device drivers with different protocols for the IO modules of the real chip (see Fig. 4 for a general model of a computing resource). They are able to cope with standard protocols as well as proprietary ones.

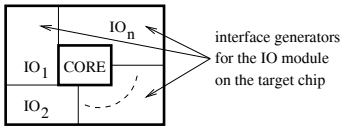

Figure 4. General model of a computing resource

The internal structures of the object-oriented computing resource models are created using polymorphism, class inheritance and object composition which provide a flexible modeling technique and allow i) the reuse of existing objects, and ii) easy creation of new computing resource models. Figure 5 shows part of the hierarchical class tree where the processor's and DSP's IO facilities are modeled.

- On the top layer the *abstract description* describes com-

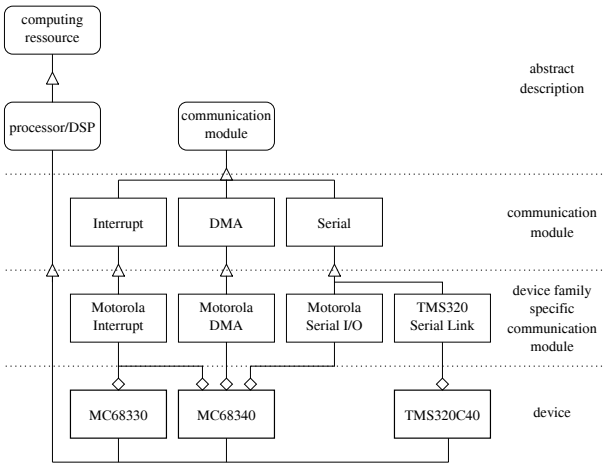

Figure 5. Object-oriented IO modeling of processor/DSP computing resources

mon properties of computing resources and communication modules (e.g., each communication module has a parameter for transfer rate).

- The *communication module* layer defines classes of IO types (e.g., Interrupt, Serial, etc.) with specific properties, e.g., which parameters are necessary for serial links.
- The classes on the layer *device family specific communication module* represent the already mentioned IO objects containing the code generators. At the moment they use parameterized templates which are stored in the object database and are automatically configured according to the communication requirements.
- On the *device* layer the processors and DSPs are compositions of IO objects.

To connect a FPGA and a processor an additional feature of the IO objects is used. Each IO object may have the ability to generate a dedicated hardware interface in VHDL for an FPGA. This interface enables the communication between FPGA and the processor's IO module and is inserted automatically into the final FPGA code.

To consider the whole communication infrastructure and to allow the generation of optimized interfaces HASIS works in two steps:

1. Provide each computing resource object with the necessary information for code generation, e.g., which edge of the task graph will use which IO module for communication in the final target.
2. Induce the involved IO objects of each computing resource object to generate the appropriate and optimized interfaces and device drivers based on the information gathered in step 1.

Necessarily, both steps involve the modification of the task and architecture graph. In step one, the task graph is updated with interface tasks (e.g., see TG in figure 6 where two tasks X and Y are bound to computing resources TMS320C40 and XC4062 respectively) which are representatives for the device drivers to generate. Essentially, their functionality is just to copy the input to the output, i.e., the functional behavior of the task graph remains the same. But in contrast to the user specified tasks (e.g., X and Y in Fig. 6) their implementation is only just generated in step two. Computing resource objects representing FPGAs are updated in step two with attributes referencing the generated HW interfaces. These interfaces are the IO modules of the FPGAs (see Figure 6) and are composed with the bound FPGAs tasks to a main entity by the code composer in a later design phase.

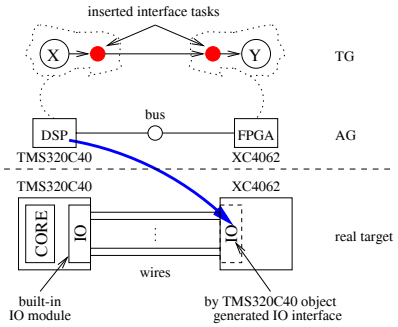

Figure 6. Generating interfaces for FPGAs

5 Case Study

Using the tool configuration of Fig. 1b) we implemented the part of a molecular dynamic problem from chemistry [10] where the inter-molecular forces of molecules are calculated using the pairlist method. Forces are not calculated between all interacting molecules but only between molecules whose distance is smaller than a given cutoff radius. The specification of this particular problem contains a task graph described by twelve task nodes and an architecture graph comprising four computing resources (nodes H, F, D and S) connected by four buses (see task and architecture graph in Fig. 2). To limit the design space we selected five processors, one DSP (TMS320C40) and eight XILINX FPGAs (xc40xx) with various speed-grades as candidates for computing resources. For the FPGAs we used task implementations with integer arithmetic to lower the task size. The "Pareto" solutions found after 500 iterations with a population size of 200 are shown in Fig. 7 where a snapshot of the design space exploration tool is presented. The points marked by two lines represent the "Pareto" points among the implementations generated by the EA. Note that three dimensions are depicted: the x-axis represents cost, the y-axis represents latency (period), and the z-axis (not shown) represents power consumption. The search space contains about $4.9 \cdot 10^{47}$ bindings (feasible and infeasible ones). Three selected "Pareto"

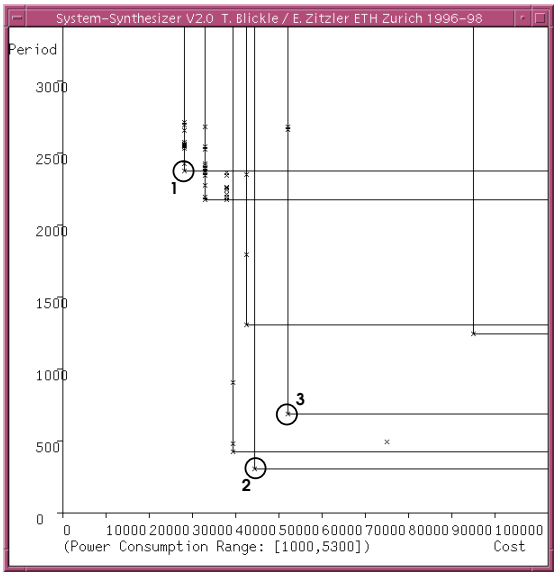

Figure 7. Snapshot of the design space exploration tool

solutions are given in Table 1.

sol.	H	F	D	S	cost	period	power
1	xc4025	DSP	xc4025	–	1	7.83	1
2	xc4025	DSP	xc4025	xc4025	1.57	1	1.08
3	xc4025	DSP	xc4062	–	1.85	2.27	1.05

Table 1. Selection of "Pareto" solutions

For each solution the type of computing resource (second to fifth column), cost, period, and power consumption are given. The figures are normalized to show the ratio between different solutions. The first solution is the cheapest and slowest and has the least power consumption (slowest FPGA speed-grade selected). The fastest but not most expensive solution is number two as there is solution tree with less power but higher cost and higher period. Between DSP and FPGAs the DSP's serial links use a blocking protocol. Between the FPGAs a proprietary bus using a handshake protocol has been selected.

6 Conclusions

The paper presents CoFrame, a modular and flexible co-design framework for heterogeneous distributed systems. The center of the approach is a pool of dynamic data structures which is analyzed, modified and refined by the application of algorithms and tools. The flexible tool configuration allows to adapt CoFrame to various control and dataflow oriented problem domains. As case study we showed the implementation of a molecular dynamic algorithm by combining a state-of-the-art multiobjective optimizer (evolutionary algorithm) and an object-oriented communication synthesis tool.

References

[1] J. Axelsson. Architecture synthesis and partitioning of real-time systems. In *Proc. of CODES/CASHE 1997, Intl. Workshop on Hardware/Software Codesign*, pages 161–165, 1997.

[2] F. Balarin, A. Jurecska, and H. Hsieh et al. *Hardware-Software Co-Design of Embedded Systems: The Polis Approach*. Kluwer Academic Press, Boston, 1997.

[3] Tobias Blickle, Jürgen Teich, and Lothar Thiele. System-level synthesis using evolutionary algorithms. *Journal on Design Automation for Embedded Systems*, 3(8):23–58, January 1998.

[4] J.T. Buck, S. Ha, E.A. Lee, and D.G. Messerschmitt. Ptolemy: A framework for simulating and prototyping heterogeneous systems. *Int. Journal of Computer Simulation*, 4:155–182, 1994.

[5] P. Chou, R. Ortega, and G. Borriello. Interface co-synthesis techniques for embedded systems. In *Proc. of the IEEE/ACM Int. Conf. on CAD (ICCAD)*, pages 280–287, San Jose, November 1995.

[6] R.P. Dick and N.K. Jha. A multiobjective genetic algorithm for hardware-software cosynthesis of distributed embedded systems. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 17:920–935, 1998.

[7] M. Eisenring and J. Teich. Domain-specific interface generation from dataflow specifications. In *Proceedings of Sixth International Workshop on Hardware/Software Codesign, CODES 98*, pages 43–47, Seattle, Washington, March 15-18 1998.

[8] M. Eisenring and J. Teich. Interfacing hardware and software. In *8th International Workshop on Field-Programmable Logic and Applications, FPL'98, Lecture Notes in Computer Science, 1482*, pages 520 – 524, Tallinn, Estonia, Aug. 31 - sept. 3 1998.

[9] R. Ernst, J. Henkel et. al. The cosyma environment for hardware/software cosynthesis of small embedded systems. *Microprocessors and Microsystems*, 20(3):159–166, May 1996.

[10] M. Gerber and T. Goessi. Parallel coprocessor architectures for molecular dynamics simulation: A case study in design exploration. In *Proc. of the 1998 IEEE International Symposium on Circuits and Systems (ISCAS)*, Monterey, June 1998.

[11] Jeffrey Horn. F1.9 multicriteria decision making. In Thomas Bäck, D. B. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*. Institute of Physics Publishing, Bristol (UK), 1997.

[12] Joern W. Janneck. Graph Type Definition Language GTDL. Technical report, ETH Zurich, 1998.

[13] A. Kirschbaum and M. Glesner. Rapid prototyping of communication architectures. In *8th IEEE Int. Workshop on Rapid System Prototyping*, pages 136–141, June 24-26 1997, Chapel Hill, North Carolina.

[14] T. Hollstein, J. Becker, A. Kirschbaum and M. Glesner. HiPART: A new hierarchical semi-interactive hw/sw partitioning approach with fast debugging for real-time embedded systems. In *Proceedings of Sixth International Workshop on Hardware/Software Codesign, CODES 98*, pages 29–33, Seattle, Washington, March 15-18 1998.

[15] Ross B. Ortega, Luciano Lavagno and Gaetano Borriello. Models and methods for hw/sw intellectual property interfacing. In *NATO ASI on System-level Synthesis*, 1998.

[16] B. Lin, S. Vercauteren, and Hugo De Man. Constructing application-specific heterogeneous embedded architectures for custom HW/SW applications. In *33rd Design Automation Conference*, pages 521–526, June 3-7, 1996.

[17] Ross B. Ortega and Gaetano Borriello. Communication synthesis for distributed embedded systems. In *Proc. of the IEEE/ACM Int. Conf. on CAD (ICCAD) 98*, pages 437–444, San Jose, California, November 8-12 1998.

[18] Gaetano Borriello Pai Chou, Ross B. Ortega. The chinook hardware/software co-synthesis system. In *8th International Symposium on System Synthesis*, pages 22–27, sept 13-15 1995.

[19] Mani B. Srivastava and Robert W. Brodersen. SIERA: A unified framework for rapid-prototyping of system-level hardware and software. *IEEE Transactions on Computer-Aided Design*, 14(6):676–693, June 1995.

[20] J. Teich, T. Blickle, and L. Thiele. An Evolutionary Approach to System-Level Synthesis. In *Proc. of Codes/CASHE'97 - the 5th Int. Workshop on Hardware/Software Codesign*, pages 167–171, Braunschweig, Germany, March 1997.

[21] Jürgen Teich, Eckart Zitzler, and Shuvra S. Bhattacharyya. 3d exploration of software schedules for dsp algorithms. In *7th International Workshop on Hardware/Software Codesign (CODES'99)*, May 1999. to appear.

[22] F. Vahid and L. Tauro. An object-oriented communication library for hardware-software codesign. In *Proc. of Codes/CASHE'97 - the 5th Int. Workshop on Hardware/Software Codesign*, pages 81–86, Braunschweig, Germany, March 1997.

[23] Wayne H. Wolf. An architectural co-synthesis algorithm for distributed, embedded computing systems. *IEEE Trans. on VLSI*, 5(2):218–229, June 1999.

[24] Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. Technical Report 70, InstituteTIK,, ETH Zurich, February 1999. submitted to Evolutionary Computation Journal.

[25] Eckart Zitzler and Lothar Thiele. An evolutionary algorithm for multiobjective optimization: The strength pareto approach. Technical Report 43, Institute TIK, ETH Zurich, Switzerland, May 1998.

[26] Eckart Zitzler and Lothar Thiele. Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation*, 1999. to appear.