Conflicting Criteria in Embedded System Design

Michael Eisenring, Lothar Thiele, and Eckart Zitzler

Computer Engineering and Networks Laboratory
Swiss Federal Institute of Technology Zurich
Gloriastrasse 35, 8092 Zurich, Switzerland

February 7, 2000

Abstract

The design of complex embedded systems involves the simultaneous optimization of several often competing objectives. Instead of a single optimal design, there is rather a set of alternative trade-offs. The paper describes the involved issues and proposes a methodology to cope with the different sources of heterogeneity in embedded system design. This combination of a design framework, new hybrid evolutionary optimization algorithms and synthesis procedures is explained using examples from architecture, interface and software design.

1 Introduction

The heterogeneity of today's embedded systems faces developers and engineers with new problems when it comes to specifying, simulating, designing, and optimizing such complex systems. Implementations are typically comprised of software programmable components, programmable or dedicated hardware components (e.g., FPGAs, ASICs), communication and memory subsystems. For system optimization, the intricate relationship between an embedded computer and its environment must be taken into account. Different models of computation and different system objectives must be combined.

The design of embedded systems is particularly driven by cost vs. benefit trade-offs. As in many other areas in engineering, the optimization involves the simultaneous consideration of several incomparable and often competing

objectives (cost, power dissipation, reliability, etc.). Therefore, one of the major challenges in embedded system design is to cope with the intrinsic heterogeneity, concerning their implementation and specification as well as their requirements and objectives, see e.g. [1, 2].

The complexity of the design problem becomes apparent when considering the huge design space of possible implementations on the one hand and the many conflicting design goals on the other. Automated design tools are necessary in order to handle the complexity of today's systems, and to support the designer in finding the trade-off which best fits the market requirements. The paper is devoted to a methodology for the design of embedded systems by which these complex problems can be handled and has the following major components:

- A model based design methodology that allows a flexible and modular combination of tools devoted to the solution of sub-problems in the whole design path.
- Hybrid optimization algorithms which allow a multi-objective design space exploration. They are characterized by a combination of problem-specific single-objective optimization algorithms for solving sub-problems and an evolutionary multi-objective optimization algorithms for design space exploration.
- An object-based representation of architecture components such as microprocessors and programmable hardware units which enables fast design exploration and interface generation under communication constraints.

The issues involved in multi-objective optimization of embedded systems are exemplified using two design studies, namely architecture and software synthesis.

2 Design Automation for Embedded Systems

The consideration of conflicting design objectives leads to the concept of design space exploration on different abstraction levels. For example, alternative system architectures are evaluated for architecture synthesis based on early estimates of the objective functions. Therefore, tools may be executed in different orders to evaluate and explore various target implementations considering important aspects like communication overhead and implementation speed. These methods act on sets of alternative implementations stored in the repository and optimize them concurrently. Consequently, a *model based design methodology* does not propose a fixed sequence of steps

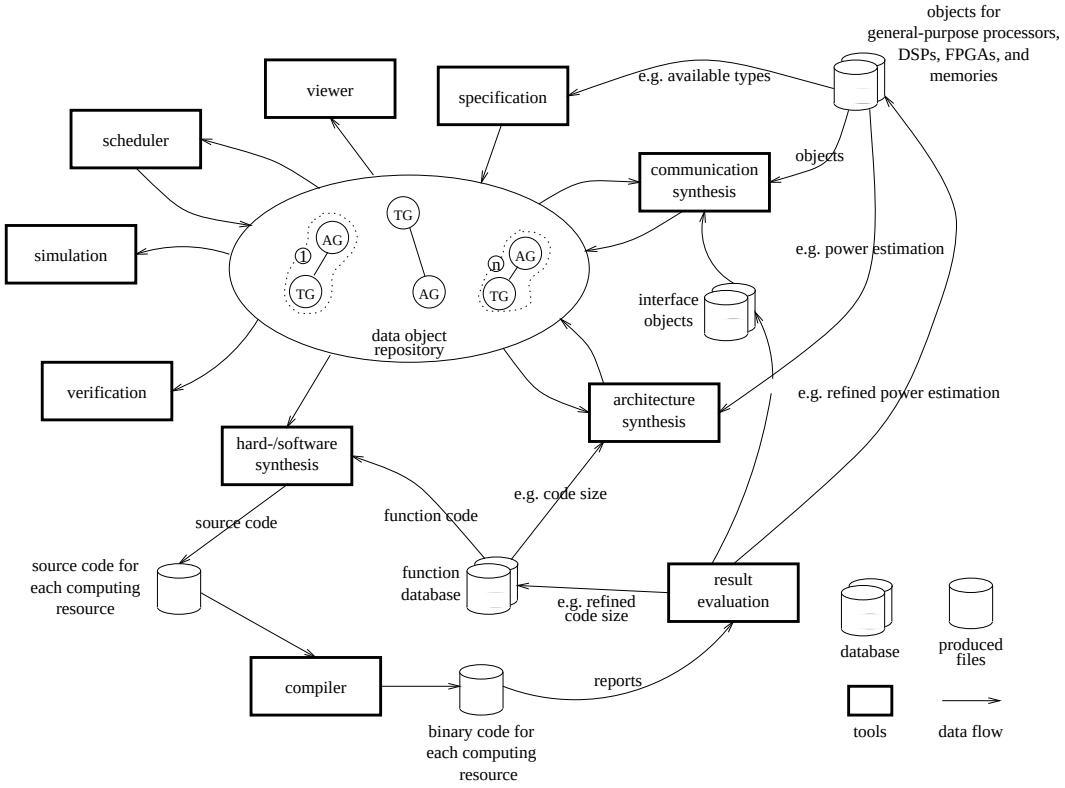


Figure 1: Model based design methodology

to refine the specification, instead providing a repository of dynamic data objects surrounded by tools working on these data objects, see Figure 1.

Specification tools allow the user to specify task and architecture graphs, including their mapping relations. In addition, user specified constraints on the application domain can be specified.

Tools for *architecture synthesis* seek for optimal implementations of a given behavior considering various user specified constraints such as task deadlines and implementation cost. The tools have access to different databases providing estimation data about task functions and computing resources (general-purpose processors, DSPs and FPGAs) including their communication properties. This exploration may also use *scheduler* tools for task and communication scheduling.

Communication synthesis tools establish the necessary communication infrastructure on the target architecture according to the task execution model and generate interface circuitry and software drivers based on the object database of computing resources.

Hardware/software synthesis tools gather the required information for each programmable component and produce appropriate source code com-

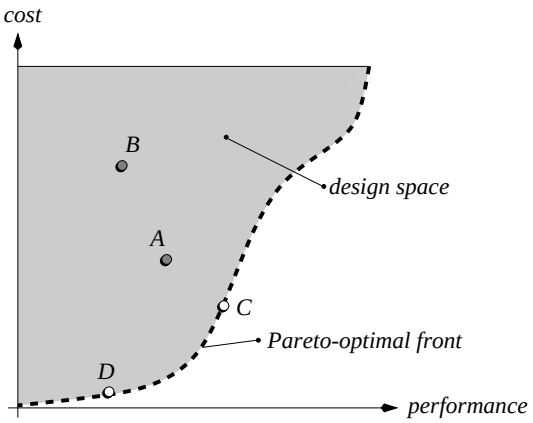


Figure 2: Illustration of the concept of Pareto optimality.

prising the generated device drivers and tasks described within a function library. These sources are compiled into hardware or software binaries by using *compiler* tools. *Result evaluation* tools help to assess the compiled code and provide information to refine the estimations within the databases.

The central *data object repository* comprises the user specification object and populations of prospective solution objects on various abstraction levels. For example, in case of architecture synthesis, each data object consists of a behavior specification in form of a *task graph (TG)* of communicating tasks, interfaces and intermediate memories, an *architecture graph (AG)* of connected computing resource objects, and a *mapping function (M)* relating TG and AG.

3 Trade-offs between Multiple Design Criteria

Let us assume we want to design an embedded system with regard to the two criteria performance and cost. These two objectives are generally competing and cannot be optimized simultaneously: high-performance architectures substantially increase cost, while cheap architectures usually provide low performance. This makes clear that a new notion of optimality is required in the presence of objective conflicts: what is the optimal trade-off of cost versus performance?

3.1 Pareto's Concept

Vilfredo Pareto [3] studied this question as early as the end of the last century. He recognized that, similar to the case when only a single criterion is considered, still some solutions are better than others when several objectives are involved. This fact is illustrated in Figure 2 where a subset of all possible designs is plotted in the space defined by the two criteria cost and performance. Comparing design A and B, A is obviously preferable to B as it provides higher performance and causes lower cost; we say A *dominates* B [4]. Generally stated, a solution is superior to (or dominates) another if it is at least as good in all criteria and better in at least one criterion. Accordingly, an optimal solution can be defined as one which is not dominated by any other solution in the design space; design C in Figure 2 represents such a system. However, in contrast to the single-objective case, where two solutions are equal or either is better, different designs may not be comparable as, e.g., C and D. While the former offers better performance, the latter achieves lower cost. As a consequence, there is no single optimal solution but rather a set of optimal trade-offs. They are called *Pareto optimal* and characterized by the fact that no improvement in any criterion can be achieved without causing a degradation in at least one other criterion. In Figure 2, the set of Pareto-optimal solutions is represented by the dashed curve, also denoted as *Pareto-optimal front*.

3.2 Decision Making

As long as no further information is given, none of the Pareto-optimal solutions can be said to be superior to the others. Thus, a decision making process is necessary in which the designer has to select one of the Pareto-optimal designs. One way to accomplish this is, for instance, to rank the design goals according to their relative importance. However, often there is a need to find or approximate the Pareto-optimal front as a basis for the selection process. The knowledge about the alternative trade-offs helps to choose a solution which best fits the market requirements. Therefore, computer-based methods for generating Pareto-optimal designs can support the designer in handling multiple conflicting design criteria.

3.3 Optimization through Evolution

Evolutionary algorithms (see Figure 3) are especially suited to this type of problem as they are capable of sampling large and complex search spaces for multiple Pareto-optimal solutions in parallel. The goal is to drive the evolution process towards the Pareto-optimal front such that i) the distance to it is minimized and ii) a wide range of different solutions is found. For many complex applications, however, it may not be possible to generate

Pareto-optimal solutions within reasonable time limits due to the intractably large search space.

Another issue that is related to the complexity of an application is the incorporation of domain knowledge by means of problem-specific search algorithms. On the one hand, evolutionary algorithms are typically used for general global design space exploration with respect to all objectives. Several programming packages are available by which evolutionary algorithms can be easily adapted to various problem domains. On the other hand, problem-specific local search methods can solve sub-problems regarding one of the objectives. Combining both global and local search into a hybrid approach has the potential to exploit the complementary advantages of both types of algorithms, which, in turn, increases the effectiveness of optimization.

In the following section two examples are given for a design space exploration on the basis of hybrid evolutionary algorithms.

4 Multiobjective Design Examples

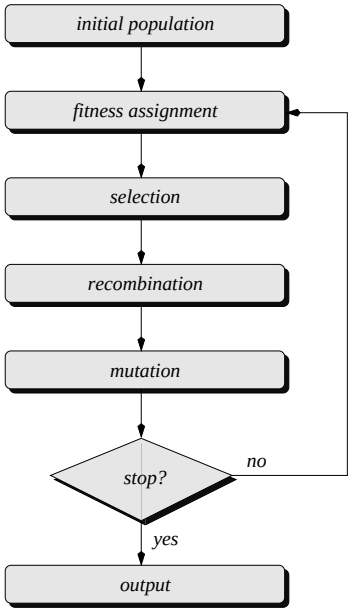
4.1 Architecture Synthesis

4.1.1 Design Problem

The goal of the architecture synthesis is to find Pareto-optimal target architectures for a given behavior considering objectives like power consumption, cost and execution time. This synthesis includes:

- the *allocation* of appropriate electronic components, i.e., the number and types of general-purpose processors, DSPs, FPGAs, buses and memories,
- the *mapping* of task graph nodes to electronic components,
- the *scheduling* of tasks on computing resources, i.e., determination of the sequence of task executions, and
- *communication synthesis* that provides the communication infrastructure on the target architecture to connect the communicating tasks and memories.

The behavioral description of a system is typically given in form of a *task graph TG* where the nodes stand for functional objects like tasks, interfaces, intermediate memories or processes, and the edges model connections between them. The structural specification of the class of all possible implementations is given in the form of an *architecture graph AG* whose nodes represent objects like general or special-purpose processors, ASICs, buses, and memories. The edges model connections between them. In addition, a set models all *feasible mappings M* between the nodes of the task graph



Evolutionary algorithms simulate natural evolution by an iterative computation process containing the following steps:

Initial population: At the beginning the population is initialized with randomly generated solutions, where each solution is appropriately encoded.

Fitness assignment: Each solution in the population is evaluated with respect to the optimization task and assigned a scalar value, the so-called *fitness*.

Selection: Low-quality solutions are removed from the population, while high-quality solutions are reproduced. The goal is to focus the search on promising portions of the search space.

Recombination: The encoded information of existing solutions (parents) is recombined in order to create new solutions (offspring).

Mutation: Some of the solutions are modified by changing small parts of the associated representation.

Termination: The simulation is stopped after a certain number of loop iterations have been performed or another condition is fulfilled.

Output: At the end, the best individual(s) in the final population or found during the entire evolution process is the outcome of the evolutionary algorithm .

Figure 3: Flow chart of an evolutionary algorithm.

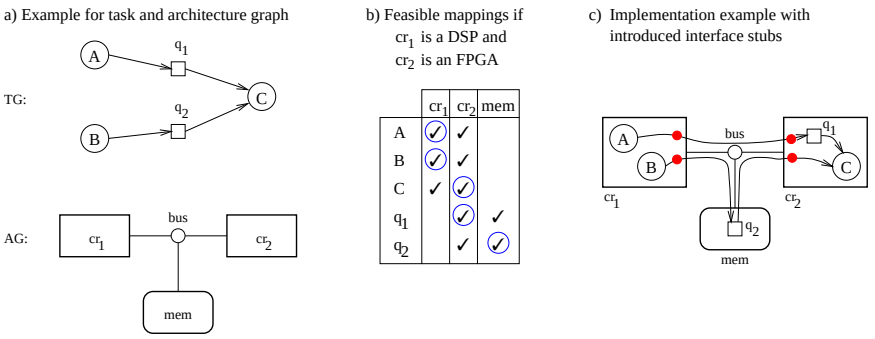


Figure 4: Architecture synthesis

and the nodes the architecture graph. Annotations to the elements of these graphs and sets can be used to specify additional quantities like deadlines, task invocation data, communication demands, estimates on communication and computation time or power consumption. In the above terms, allocation selects structural objects from the architecture graph, binding selects a subset of the feasible bindings and scheduling assigns to each edge a starting time.

Unfortunately, allocation, final mapping (binding), scheduling, and communication synthesis are not independent. For example, Figure 4a) shows a task graph (tasks A, B, and C, intermediate memories q_1 and q_2) and the structure of an architecture graph (computing resources cr_1 and cr_2, memory mem). The allocation of components (e.g, cr_1 is a DSP and cr_2 is a FPGA) restricts the set of feasible mappings, see Figure 4b). According to the table, the whole task graph might be implemented on the FPGA. But depending on the tasks and the intermediate memory sizes the FPGA may be very expensive. However, if several computation units are used, interfaces between them are necessary. The design space grows enormously as different physical communication styles (e.g., DMA, interrupt driven, dedicated links) and protocols (e.g., blocking, non-blocking) may be used between computation units. For example, Figure 4c) shows the implementation of the selected mapping in Figure 4b), where the required interface stubs are indicated with a filled circle. Additionally, as cr_1 is a sequential computing resource the task execution order for the tasks A and B must be determined.

4.1.2 Solution Strategy

The hybrid evolutionary algorithm incorporates problem-specific assessment procedures, which evaluate the current implementation based on estimations, and both heuristics and exact algorithms for optimizing sub-goals. The major components of the architecture synthesis can be described as follows, see [5].

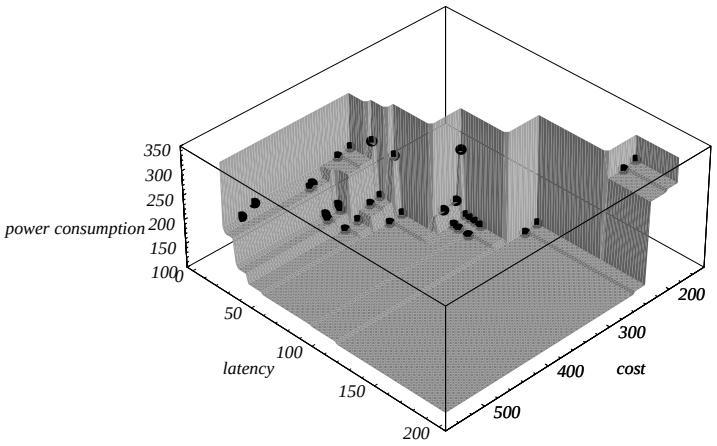


Figure 5: Three-dimensional trade-off front for the video codec application.

- The schedule is computed using a list-scheduling heuristic which incorporates loop pipelining and memory optimization.
- An allocation is represented by a bit vector which defines for each structural object whether it is selected or not. In order to reduce the number of infeasible solutions, allocations are partially repaired by a heuristic. For the same reason, bindings are not encoded directly by a bit vector, but rather indirectly by several lists.
- Intricate functions which depend on the allocation, binding, and scheduling are used to estimate the various objectives.
- The HASIS [7] synthesis system allows the generation of interfaces between hardware and software units, and supports a variety of processor families, communication channels and protocols. It provides a hybrid approach between library based interface instantiation [8] and synthesis from formal specifications [9] on the basis of object-oriented classification of architecture graph components.
- The SPEA-algorithm for multi-objective optimization has been used for the design space exploration, see [6].

4.1.3 Video Codec Example

As an example, we consider the architecture synthesis of a video codec, based on the H.261 standard, in the following. The specification of the system including task graph, architecture graph, binding space, communication specifications, etc. can be found in [5].

The trade-off front produced by the hybrid evolutionary algorithm is shown in Figure 5, which well reflects the conflicts between the three objectives; cost, latency, and power consumption. The cheapest solution (the rightmost point) provides the lowest performance at maximum power dissipation (cost: 180, latency: 166, power consumption: 299). The target consists of a general-purpose RISC processor, two I/O devices, and a slow memory module connected by a slow bus. The components use a shared memory approach where the RISC processor is the bus-master.

In contrast, the fastest solution (the leftmost point), that includes several specialized computing resources and a fast bus, causes the maximum cost at medium power dissipation (cost: 520, latency: 22, power consumption: 218). For fast data transmission the components use built-in DMA controllers working in a cycle-stealing mode. A good compromise solution could be the one represented by the point $(360, 42, 154)$: low power dissipation (154) and good performance (42) at medium cost (360). It differs from the fastest solution in that it uses a slower bus and a subtraction/adder module instead of a faster digital signal processor. Again, the components use a simple shared memory approach for communication.

4.2 Software Synthesis

4.2.1 Design Problem

Consider the following scenario: the architecture and the partitioning of an embedded system to be designed have been determined, and a particular part of the overall system is to be implemented on a digital signal processor. Similar to the other levels of abstraction, this specific behavior is described by means of a formal specification, in this case synchronous data flow (SDF) graphs. An SDF graph is a refined task graph where for each subtask it is specified how many data entities are produced and consumed per communication channel. An example for an SDF graph is depicted in Figure 6a), consisting of two communicating subtasks A and B where A produces two data entities per execution and B consumes three.

The problem we address here is the automatic generation of a software implementation for the particular digital signal processor from the given SDF specification. It is part of the hardware/software synthesis depicted in Figure 1. In this context, two subproblems have to be solved:

- A valid sequence of subtask executions must be found such that the number of data entities per communication channel is the same before and after the execution of the sequence. This guarantees that the overall algorithm can potentially run forever on the digital signal processor without deadlocking. In Figure 6b), the execution of such a sequence is illustrated.

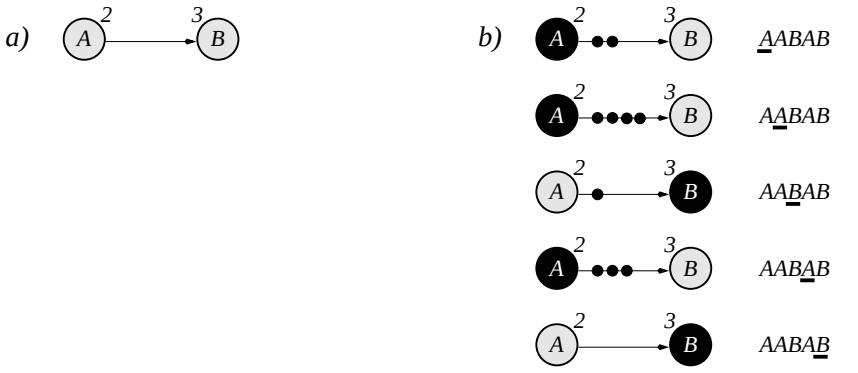


Figure 6: A simple synchronous data flow graph (a) and the execution of the subtask sequence $AABAB$ (b).

- Assuming that optimized assembly code is given for each subtask, an overall program must be generated from the distinct code fragments. There are several ways to implement a sequence of subtask executions. For each execution of a subtask the corresponding assembly code can be copied into the final program (inlining). Alternatively, subtasks may be implemented as subroutines where each execution is realized via a subroutine call. A third possibility is the incorporation of software loops by which identical parts of the program can be grouped. Combinations of these techniques are possible also.

The optimization criteria taken into account are: data memory requirement, program memory requirement, and execution time. The amount of data memory that is necessary to store the data entities exchanged between the subtasks is solely determined by the sequence of subtask executions. In the example depicted in Figure 6b), at most four data entities must be stored. The execution time is influenced by the time needed per subtask execution and by the way the final program code is structured. The usage of subroutines and/or loops causes an overhead in execution time due to the necessary context switches, branch condition checks, etc. Moreover, this overhead also affects the total size of the overall program. While subroutines and loops save program memory, directly copying subtask code into the program increases its size.

4.2.2 Solution Strategy

In order to approximate the Pareto-optimal front of software implementations with regard to the above optimization criteria, a hybrid approach was chosen:

- Before the search, the minimum number of appearances per subtask that each valid execution sequence must contain is determined by a depth-first search algorithm.
- On the basis of this information, an evolutionary algorithm explores different sequences of subtask executions with regard to the three objectives. A repair heuristic is incorporated that prevents the generation of invalid sequences.
- A problem-specific deterministic algorithm is used during the evolutionary search, which computes an optimal looping for a given sequence of subtask executions.
- An evaluation module estimates the data and program requirements as well as the execution time required by a particular software implementation, taking the specific characteristics of the target processor into account.

4.2.3 CDtoDAT Example

As an example, a sample rate conversion system, where a compact disc player is connected to a digital audio tape, is considered here [6]. The trade-off fronts obtained for this application for two different target processors (Motorola DSP56k and TI TMS320C40) are depicted in Figure 7. The rightmost points represent software implementations that use neither software loops nor subroutine calls. These solutions do not produce any overhead in the execution time dimension, but need a maximum of program memory. In contrast, the implementations represented by the leftmost points make excessive use of looping and subroutines which leads to minimal program memory requirements, at the expense of a maximum execution time overhead.

It is remarkable how software loops interfere with data memory requirement. Software loops introduce an indirect trade-off between program and data memory requirements because looping depends on the execution sequence of the subtasks: some execution sequences which can be looped well may have high data memory requirements and vice versa. This trade-off is reflected by the variations in data memory requirements and illustrated by the points that are close to each other regarding program memory and execution time, but strongly differ in the data memory dimension.

5 Conclusions

Inspite of the advances in understanding the major issues involved in embedded system design, there are several open questions closely related to the described approach.

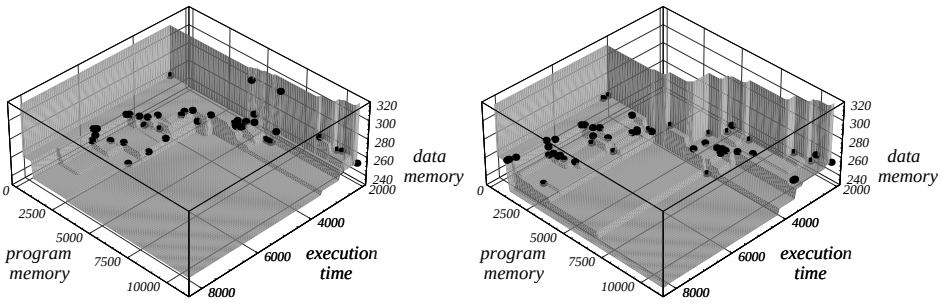


Figure 7: Trade-off fronts achieved for the Motorola DSP56k (left) and the TI TMS320C40 (right) using a hybrid evolutionary algorithm.

The classical approach to design and optimization is characterized by the fact that decisions about the importance of the involved design criteria are made prior to the optimization. The results shown in the paper proceed in the exactly opposite direction, i.e., at first the (multi-objective) optimization and design space exploration is performed and then decision are made. Experience shows that in the presence of complex design spaces the number of different Pareto-optimal solutions can be overwhelming. Therefore, new intermediate possibilities are looked for that combine the above described extreme possibilities.

Furthermore, the complexity of the whole design trajectory implies a hierarchical approach. For example, one may at first design sub-components that are then combined to the overall system or one may decompose the whole system into different levels of abstraction that are handled using the conventional top-down/bottom-up approach. It would be of major interest to know whether these hierarchical approaches must be reconsidered in the presence of multiple optimization criteria.

References

[1] F. Balarin et al., *Hardware-Software Co-Design of Embedded Systems: The Polis Approach,* Kluwer Academic Press, Boston, June 1997.

[2] R. K. Gupta, *Co-Synthesis of Hardware and Software for Digital Embedded Systems*, vol. 329, Kluwer Academic Publishers, Boston, August 1995.

[3] V. Pareto, *Cours D'Economie Politique*, vol. 1, F. Rouge, Lausanne, 1896.

[4] R. E. Steuer, *Multiple Criteria Optimization: Theory, Computation, and Application*, Wiley, New York, 1986.

[5] T. Blickle, J. Teich, and L. Thiele, "System-level synthesis using evolutionary algorithms," *Design Automation for Embedded Systems*, vol. 3, no. 1, pp. 23–58, 1998.

[6] E. Zitzler, J. Teich, and S. S. Bhattacharyya, "Multidimensional exploration of software implementations for DSP algorithms," *VLSI Signal Processing Systems*, 2000, To appear.

[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*, Seattle, Washington, March 15-18 1998, pp. 43–47.

[8] 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*, Braunschweig, Germany, March 1997, pp. 81–86.

[9] P. Chou, R. Ortega, and G. Borriello, "Interface Co-Synthesis Techniques for Embedded Systems," in *IEEE/ACM International Conference on Computer-Aided Design, Santa Clara, CA*, November 1995, pp. 280–287.