Multi-Objective Topology Optimization for Networked Embedded Systems

Thilo Streichert and Christian Haubelt and Jürgen Teich
Computer Science 12
University of Erlangen-Nuremberg, Germany
Email: streichert,haubelt,teich@cs.fau.de

***Abstract*— In this paper, a new methodology is presented for topology optimization of networked embedded systems as they occur in automotive and avionic systems and partially in wireless sensor networks. By introducing a model which is (1.) suitable for heterogeneous networks with different communication bandwidths, (2.) modeling of routing restrictions, and (3.) flexible binding of tasks onto processors, current design issues of networked embedded systems can be investigated. On the basis of this model, the presented methodology firstly allocates the required resources which can be communication links as well as computational nodes and secondly binds the functionality onto the nodes and the data dependencies onto the links such that no routing restrictions will be violated or capacities on communication links will be exceeded. By applying Evolutionary Algorithms, we are able to consider multiple objectives simultaneously during the optimization process and allow for a subsequent unbiased decision making. An experimental evaluation as well as a demonstration of a case study from the field of automotive electronics will show the applicability of the presented approach.**

I. Introduction

Embedded networks that can be found, e.g. in automotive systems, nowadays consist of up to 100 Electronic Control Units (ECUs) which are connected via different types of shared buses. Several communication standards like LIN [10], CAN [4], FlexRay [5] or TTP [15] combined with lots of design alternatives concerning the computational nodes, increase the design complexity of the entire networked embedded system. Moreover, the networked embedded system executes functionality which is typically distributed and consists of communicating processes statically bound onto computational nodes in the network. A system-level designer, hence, has to take the decision about which computational and communication resources are required and where to execute tasks in the network such that no overload occurs on nodes and the capacity of communication links is not exceeded. Additionally, all these decisions have to be taken by respecting different constraints and objectives, like minimization of monetary costs, power consumption or maximizing fault-tolerance.

In this paper, we consider such networks that consist of computational nodes which are able to execute a certain amount of software load, and links with a certain capacity for the communication demand between the functions. Our methodology requires a so-called *architecture graph* [3] containing all available resources. Out of these available resources the resources for the final system are selected and the functionality represented by a *problem graph*, introduced later on, is bound onto the selected network nodes. Moreover, the messages between the functions are bound to the communication links. By respecting multiple objectives, our methodology determines a set of so-called *Pareto-optimal* solutions that allows for an unbiased decision making.

Several approaches exist targeting a similar problem which is commonly referred to as *design space exploration* of embedded systems. Unfortunately, as these approaches are destined for SoC designs, no straightforward extensions exist for exploring the implementation alternatives of networked embedded systems as they occur in automotive, avionic, or wireless sensor networks.

For signal processing architectures, SPADE (System-level Performance Analysis and Design space Exploration) [9] is a tool for performance analysis. This tool is incorporated by Artemis (Architectures and Methods for Embedded Media Systems) which explores the design space [13].

Another framework, called MILAN (Model-based Integrated simuLAtioN), is a design space exploration tool that works at different levels of abstraction [12]. Hierarchical data flow graphs including alternatives for application specification as well as an architecture template will be defined and explored at different levels of detail before simulative evaluation.

Thiele et al. [14] propose a design space exploration methodology based on Evolutionary Algorithms for packet processing applications, called EXPO.

Kianzad and Bhattacharyya propose a framework called CHARMED (Co-synthesis of HARdware-software Multi-mode EmbeddeD systems) [7] for the automatic design space exploration for periodic multi-mode embedded systems.

Balarin et al. [1] propose Metropolis, a design space exploration framework which integrates tools for simulation, verification, and synthesis.

All these tools have in common that they either do not consider communication at all or assume restricted binding conditions by requiring explicit communication modeling which is prohibitive in networked embedded systems. Here,

we will present a strategy for solving a *multi-commodity* or *multi-concurrent* flow [6] problem together with the binding of functionality onto computational nodes in the network.

The paper is structured as follows: While the next section introduces the network system model, gives an example and reasons for the chosen model, Section III explains our methodology for topology optimization of networked embedded systems. An evaluation of the proposed strategy will be given in Section IV before concluding in Section V.

II. Network System Model

The input to the topology optimization framework is a so-called *specification graph*. In this framework, we strictly separate behavior and structure:

Definition 1 (Specification Graph): A *specification graph* $g_\mathrm{s} = (g_\mathrm{p}, g_\mathrm{a}, E_\mathrm{m}, M)$ consists of a *problem graph* g_p, an *architecture graph* g_a, *mapping edges* E_m, and a set of message types M.

Problem and architecture graph can be defined formally:

Definition 2 (Problem Graph): A *problem graph* $g_\mathrm{p} = (V_p, E_p)$ consists of vertices $v \in V_p$ and edges $e = (v_i, v_j) \in E_p \subseteq V_p \times V_p$.

Definition 3 (Architecture Graph): An *architecture graph* $g_\mathrm{a} = (V_a, E_a)$ consists of vertices $v \in V_a$ and edges $e = (v_i, v_j) \in E_a \subseteq V_a \times V_a$.

The problem graph g_p represents the set of applications to be realized by the implementation. Vertices represent processes and edges represent data dependencies between the processes. The architecture graph $g_\mathrm{a}(V_\mathrm{a}, E_\mathrm{a})$ models the template for the architecture of the system. As mentioned before, the architecture graph consists of all available resources. During the topology optimization phase, a subset of these resources will be selected (see Section III) for implementation. Vertices $v_\mathrm{a} \in V_\mathrm{a}$ represent resources and the connections of resources are modeled by edges $e_\mathrm{a} \in E_\mathrm{a}$. Finally, the mapping edges $e_\mathrm{m} \in E_\mathrm{m}$ relate vertices of the problem graph g_p with vertices of the architecture graph g_a. A mapping edge $e_\mathrm{m} \in E_\mathrm{m}$ indicates the possible implementation of a process on the corresponding resource. An example of a specification graph is shown in Fig. 1. Gray nodes connected via directed edges represent functions with their data dependencies and white nodes represent resources connected via directed edges. The dashed edges in Fig. 1 represent mapping edges. In our model, we distinguish a finite number of message types. Each message type $m \in M$ corresponds to a communication protocol in the networked embedded system.

Definition 4 (Message Type): M denotes a finite set of message types $m_i \in M$.

In networks with links supporting different bandwidth protocols and bandwidths, it is crucial to distinguish different demands. Assume a certain amount of data has to be transferred between two nodes in a network. Between these nodes are two types of network, one which is dedicated for data transfer and

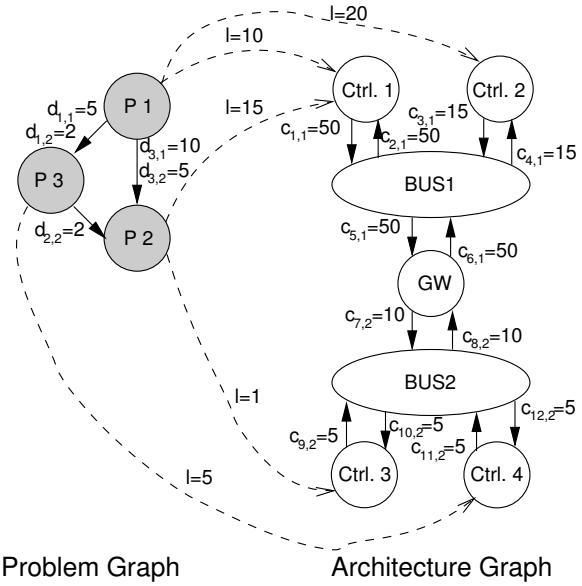

Fig. 1. Model for the exploration of network topologies: Edges in the problem graph are annotated with demands. Edges in the architecture graph are annotated with capacities. Note that capacities and demands which are equal to zero or infinity, resp., are not shown.

supports multi-cell packages and one which is dedicated for, e.g., sensor values and therefore has a good payload/protocol ratio for one word messages. In such a case, the data which has to be transferred over two different networks would cause a different traffic in each network. Hence, we associate with each edge $e \in E_p$ so-called *demand values* which represent the required bandwidth when using a given message type or kind of network, respectively. An example for a network consisting of heterogeneous, multiple protocols can be found in automotive systems, where CAN-buses of different speed grades are connected to, e.g, a LIN- or MOST-bus. If messages have to be transferred between nodes connected to these different bus systems, a gateway has to be passed for adapting the messages to the corresponding network type.

Definition 5 (Demand): With each pair $(e_i, m_j) \in E_\mathrm{p} \times M$, we associate a real value $d_{i,j} \in \mathcal{R}_0^+$ (possibly ∞, if the message type cannot occur) indicating the *demand* for communication bandwidth by the two adjacent processes.

Exemplarily, Fig. 1 shows a problem graph consisting of three nodes with three demands. While the demand between $P1$ and $P2$ as well as the demand between $P1$ and $P3$ can be routed over all two network types ($|M| = 2$), the demand between $P2$ and $P3$ can be routed only over a network that can transfer message type m_2. This will be expressed by setting $d_{2,1}$ for edge $e_2 = (P2, P3)$ between $P2$ and $P3$ to ∞. On the other hand, the supported bandwidth is modeled by so-called *capacities* to each message type $m \in M$ associated with edges $e \in E_a$ in the architecture graph.

Definition 6 (Capacity): With each pair $(e_i, m_j) \in E_\mathrm{a} \times M$, we associate a real value $c_{i,j} \in \mathcal{R}_0^+$ (possibly 0, if the message

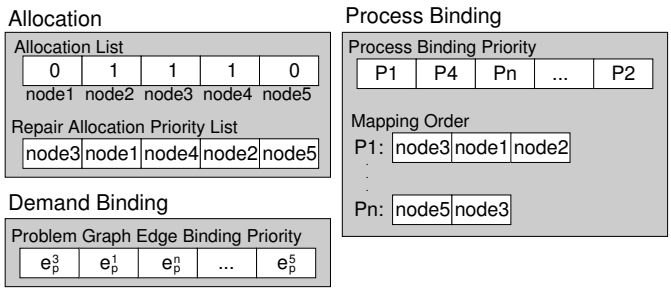

Fig. 2. The chromosome structure for the allocation, the process binding, and the demand binding are shown.

type cannot be routed over e_i) indicating the *capacity* on a link e_i for message type m_j. For each edge $e_i \in E_a$, exactly one capacity c_i is greater than 0.

Fig. 1 shows a network consisting of four computational nodes (Ctrl.1,...,Ctrl.4), one gateway (GW) and two buses. While $BUS1$ can transfer the message type $m1$, $BUS2$ can handle message type $m2$. The gateway can convert a message of type $m1$ to a message of type $m2$ and vice versa. Note that only capacities $c > 0$ and demands $d < \infty$ are shown in this figure. In our model, we assign exactly one capacity with $c > 0$ to each edge $e \in E_\mathrm{a}$ in the architecture graph and at least one demand with $d < \infty$ to the edges $e \in E_\mathrm{p}$ in the problem graph. Depending on the type of capacity, a demand of the corresponding type can be routed over such an architecture graph edge. With this extension, it is possible to limit the routing possibilities, and moreover, to assign different demands to one problem graph edge.

III. Topology Optimization

From the previously described specification graph, the topology optimization framework (a) selects a subset of resources, (b) binds processes to these resources, and (c) assigns demands to a path $p = (e_1, e_2, \ldots, e_n)$ where $e_1, \ldots, e_n \in E_\mathrm{a}$ and $e_1 = (v_0, v_1), e_2 = (v_1, v_2), \ldots, e_n = (v_{n-1}, v_n)$ with $v_i \in V_\mathrm{a}$. In summary, the topology optimization framework generates solutions to the given specification by using *Multi-Objective Evolutionary Algorithms*. Basically, this is done by encoding solutions in so-called *chromosomes* (see Fig. 2). Each solution can be decoded to an implementation (the so-called *phenotype*). Our topology optimization framework makes use of the formal definition of an *implementation* as given in [3]. In our case, an implementation consists of three parts: (i) the *allocation* that indicates which elements of the problem and architecture graph are used in the implementation, (ii) the *process binding*, i.e., the set of mapping edges which defines the binding of vertices in the problem graph to components of the architecture graph, and (iii) the *demand binding* assigning a problem graph edge with its demands to a path in the architecture graph while satisfying capacity constraints.

Before defining the term *implementation* formally, we will explain the concept of activation as described in [3].

Definition 7 (Activation): The *activation* of a specification graph $g_\mathrm{s}(g_\mathrm{p}, g_\mathrm{a}, E_\mathrm{m}, M)$ is a function $\mathrm{a} : V_\mathrm{p} \times V_\mathrm{a} \times E_\mathrm{p} \times E_\mathrm{a} \times E_\mathrm{m} \mapsto \{0, 1\}$ that assigns to each edge and to each vertex the value 1 (activated) or 0 (not activated).

The task of topology optimization is to determine an implementation, i.e., an assignment of activity values to vertices and edges of the specification graph. An *allocation* α of a given specification graph g_s is the subset of all activated vertices and edges of the problem graph g_p and the architecture graph g_a, i.e., $\alpha = \alpha_\mathrm{v} \cup \alpha_\mathrm{e}$, where $\alpha_\mathrm{v} = \{v \in V_\mathrm{p} \cup V_\mathrm{a} \mid \mathrm{a}(v) = 1\}$ and $\alpha_\mathrm{e} = \{e \in E_\mathrm{p} \cup E_\mathrm{a} \mid \mathrm{a}(e) = 1\}$.

A *process binding* β_p of a given specification graph g_s is the subset of activated mapping edges E_m, i.e., $\beta_\mathrm{p} = \{e \in E_\mathrm{m} \mid \mathrm{a}(e) = 1\}$.

In order to restrict the search space, it is useful to determine the set of feasible allocations and feasible process bindings. A feasible process binding guarantees that communications demanded by the problem graph can be established in the allocated architecture. This is an important property in explicit modeling of communications. Hardly any other model known from literature exposes this property.

Definition 8 (Feasible Process Binding): Given a specification graph g_s and an allocation α, a *feasible process binding* is a process binding β_p that satisfies the following requirements:

1) Each activated mapping edge $e_\mathrm{m} \in \beta_\mathrm{p}$ starts and ends at an activated vertex, i.e., $\forall e_\mathrm{m} = (v_\mathrm{p}, v_\mathrm{a}) \in \beta_\mathrm{p} : v_\mathrm{p}, v_\mathrm{a} \in \alpha$.
2) For each activated problem graph vertex $v_\mathrm{p} \in V_\mathrm{p} \cap \alpha$, only one outgoing mapping edge $e_\mathrm{m} \in E_\mathrm{m}$ is activated, i.e., $|\{e_\mathrm{m} \in \beta_\mathrm{p} \mid e_\mathrm{m} = (v_\mathrm{p}, v_\mathrm{a}), v_\mathrm{a} \in V_\mathrm{a}\}| = 1$.
3) For each activated problem graph edge $e_\mathrm{p} \in (v_{\mathrm{p},i}, v_{\mathrm{p},j}) \in E_\mathrm{p} \cap \alpha$ with $((v_{\mathrm{p},i}, v_{\mathrm{a},i}), (v_{\mathrm{p},j}, v_{\mathrm{a},j})) \in E_\mathrm{m}$ a path p from $v_{\mathrm{a},i}$ to $v_{\mathrm{a},j}$ exists.

This definition differs from the concepts of feasible binding presented in [3] in a way that communicating processes require a *path* in the architecture graph and not a direct link for establishing this communication. This way, we are able to consider *networked embedded systems*. However, considering multi-hop communications, we have to regard the capacity of connections and data demands of communication. This step will be named *demand binding* in the following.

Definition 9 (Feasible Demand Binding): The process of *demand binding* can be expressed by using the following ILP formulation: We define a binary variable with

$$x_{i,j} = \begin{cases} 1: & \text{problem graph edge } e_{\mathrm{p},i} \text{ is bound on} \\ & \text{architecture graph edge } e_{\mathrm{a},j} \\ 0: & \text{else} \end{cases}$$

and a mapping vector $\overrightarrow{m_i} = (m_{i,1}, \ldots, m_{i,|V_\mathrm{a}|})$ for each problem graph edge $e_{\mathrm{p},i} = (v_{\mathrm{p},k}, v_{\mathrm{p},j})$ with the elements

$$m_{i,l} = \begin{cases} 1: & \text{if mapping edge } e_{k,l} = (v_{\mathrm{p},k}, v_{\mathrm{a},l}) \in \beta_p \\ -1: & \text{if mapping edge } e_{k,l} = (v_{\mathrm{p},j}, v_{\mathrm{a},l}) \in \beta_p \\ 0: & \text{else} \end{cases}$$

Then, the following two kinds of constraints exist:

- $\forall i = 1,..,|E_{\mathrm{p}}|$: $C \cdot \overrightarrow{x_i} = \overrightarrow{m_i}$, with C being the incidence matrix of the architecture graph and $\overrightarrow{x_i} = (x_{i,j},..,x_{i,|\mathrm{E_a}|})^T$.
  This constraint literally means that all incoming and outgoing flows of an architecture graph node have to be equal. If a demand producing or consuming process is mapped onto an architecture graph node, the sum of incoming flows differs from the sum of outgoing flows.
- The second constraint restricts the sum of demands $d_{i,j}$ bound onto an architecture graph edge $e_{\mathrm{a},j}$ to be less than or equal to the edge's capacity c_j, where $d_{i,j}$ is the demand of the problem graph edge $e_{\mathrm{p},i}$. $\forall j = 1..|E_{\mathrm{a}}|$: $\sum_{i=1}^{|E_{\mathrm{p}}|} d_{i,j} \cdot x_{i,j} \leq c_j$

Definition 10 (Feasible Allocation): A *feasible allocation* is an allocation α allowing at least one feasible binding β_p with a corresponding feasible demand binding β_d.

A. Chromosome Decoding

As mentioned above, the decoding can be subdivided into three parts, the *allocation*, the *process binding* and the *demand binding*. While the allocation of resources and the binding of processes are part of the decoding process introduced in [3], the demand binding which requires to solve a multi-commodity flow problem is new and explained in the following.

In Fig. 3 the flow of the decoding step is presented. First, the allocation of resources is determined by the *Allocation List* (cf. Fig. 2). The allocation is repaired using the *Repair Allocation List* such that all processes can be bound onto resources. This is done by inserting resources into the allocation regarding their occurence in the Repair Allocation List. After all processes are bound onto the architecture graph nodes using the processes' occurence order in the *Process Binding Priority List*, a path feasibility check is performed which checks whether two adjacent problem graph nodes can communicate over a path between the allocated resources. This check is implemented as a depth first search suitable for cyclic graphs. During this check, the demands and capacities on the edges in the problem graph or the architecture graph are not respected.

In the next phase, we have to perform the task of demand binding. Generally, there are two possible solutions in this problem: (1) Using an ILP solver for exact solutions or (2) using a heuristic by encoding the demand binding in the chromosome. In this paper, we will compare both approaches. By using an ILP solver, we use the allocation and process binding to formulate the ILP as presented in Def. 9. The objective of this ILP formulation is to minimize the total flow in the network: $\min(\sum_{i=1}^{|E_{\mathrm{p}}|} \sum_{j=1}^{|E_{\mathrm{a}}|} d_{i,j} \cdot x_{i,j})$. Using a chromosome encoding, a *Problem Graph Edge Priority List* is decoded. Each element in this list refers to a certain edge in the problem graph that has to be mapped onto the edges of the architecture graph. Beginning with the first element, the demand of the problem graph edge is bound onto the shortest path with sufficient capacities. All capacities along this path are reduced by the demand. Here, only the demand type is considered which corresponds to the capacity type of the architecture graph edge. The objective to be minimized corresponds to the ILP formulation. If no path with sufficient capacities can be found in the architecture graph, an error counter is increased by one. This error counter is another objective to be minimized and helps the EA to guide the search towards feasible solutions. If this error counter equals zero, a valid implementation has been found.

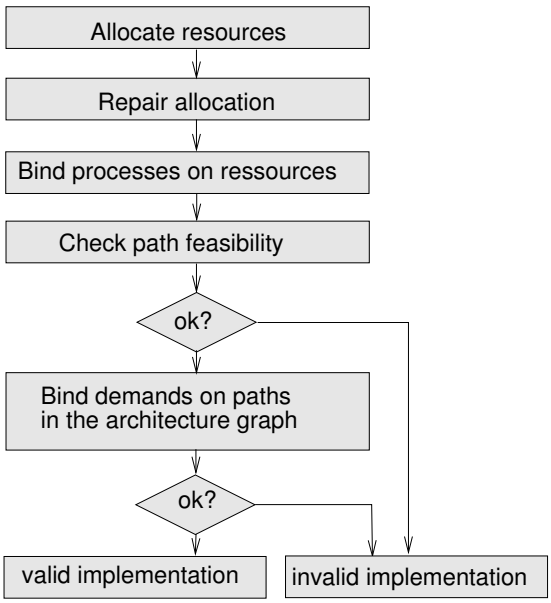

Fig. 3. Decoding process of an individual: First, an allocation is decoded and repaired such that all processes can be bound. Then, all processes are bound on resources as specified in the chromosome. If for each data dependency in the problem graph a path exists in the architecture graph, continue with the binding of demands on edges in the architecture graph with respect to capacity constraints. If all demands could be bound, the implementation is feasible.

B. Performance Evaluation and Constraint Checking

After decoding an implementation and checking its feasibility, the performance evaluation takes place. Typical evaluators are: To sum the monetary costs associated with the allocated resources resulting in an overall cost objective value. Additionally, the computational load on the network nodes as an orthogonal objective to the flow in the network can be considered. However, competing objectives allow for implementing different solutions which are all called to be *Pareto-optimal*.

As the specification graph might be annotated with arbitrary attributes, a system designer can also define additional

objectives and load customized evaluators into our topology optimization framework. In order to support different objective evaluators, all objectives are assumed to be minimized.

After performance evaluation, the constraints imposed on an implementation can be tested. For this purpose, our topology optimization framework permits dynamic loading of constraint checking algorithms, too. A constraint is specified by a lower and an upper bound. Usually, the already computed objective values can be tested by a constraint checker. If a constraint is violated, the checker is required to return one, otherwise the checker returns zero. Using this scheme, the return values of the constraint checkers can be added and only a result without constraint violation indicates a valid implementation. This sum is minimized as well, i.e., the constraint violation is treated as an additional objective value. In case of an invalid implementation, all objective values are set to infinity. Thus, the only remaining objective is to minimize the number of constraint violations. That way, the task of *system synthesis* is a multi-objective optimization problem (see e.g., [16], [8]).

IV. Experimental Evaluation

In the following experiments, we are comparing the two different implementations, one using the Evolutionary Algorithm SPEA2 [2] for binding of demands onto the resources in the architecture graph and one solving the ILP formulation from Def. 9 using LpSolve [11]. For the evaluation of these algorithms, we used applications with 10 and 20 individual demands and all mapping possibilities on a 3x3 mesh. We defined the capacities of the architecture graph edges each to 100% and produced for each number of demands three different scenarios by varying the demand sizes between 1% and 100%. For each of these scenarios, we executed three iteration runs with the EA-based and the ILP-based methodology and obtained three sets S^i_{ILP} and S^i_{EA} with $i = 1, .., 3$ containing the set of Pareto-optimal solutions after the iteration. Extracting all Pareto-optimal solutions out of the sets S^i_{ILP} and S^i_{EA} provides us a set of solutions which we assumed to contain the Pareto-optimal solutions. Therefore, these solutions were taken as a reference set R. In order to evaluate the iteration runs of our proposed heuristic and the combined approach incorporating an ILP, the shortest normalized distance $d(s)$ between the Pareto-front of the reference set R and the solutions $s \in S^i_{\mathrm{ILP}}$ or $s \in S^i_{\mathrm{EA}}$ resp. for all iteration runs and scenarios are determined:

$$d(s) = \min_{r \in R} \left\{ \left| \frac{s_{o1} - r_{o1}}{r^{max}_{o1} - r^{min}_{o1}} \right| + \left| \frac{s_{o2} - r_{o2}}{r^{max}_{o2} - r^{min}_{o2}} \right| \right\}$$

The indices $o1$ and $o2$ denote the two objectives, for a considered points, whereas, the indices max and min denote the maximal value or the minimal value of the points belonging to the reference set R.

The average distances and standard deviations in each iteration for the cases with 10 demands and 20 demands are presented in Fig. 4a) and Fig. 4b). Fig. 5a) and Fig. 5b) show the distance over the exploration time. The topology optimization has been executed on a Intel Pentium IV (2.7GHz/512MB RAM) running Linux. We can clearly see that our proposed heuristic converges faster and at the same time runs faster than the hybrid approach of an EA with an exact ILP formulation (EA/ILP).

A case study from the field of automotive applications aimed at optimizing the network topology and the binding of an adaptive light controller. In this case study, more than 100 different processes are producing, processing and consuming data. These processes need to be bound onto a network consisting of 36 sensors, 30 controllers/gateways, and 35 actuators. While the controllers could be connected with different types of buses, we connected each sensor and actuator via point-to-point (P2P) communication links to each controller in the specification graph. One objective of the topology optimization was to find solutions that minimize the total wire length of the P2P connections and the bus systems while the topology optimization on the other hand aimed at minimizing the monetary cost of the entire system which has a direct effect to the allocation of resources. Additionally, certain demands have been annotated to the edges between the functional units which have to be bound to the edges and nodes between the sensors, controllers/gateways and actuators. By combining all demand routing alternatives, binding possibilities and resource allocation options, the search space incorporated about 2^{300} possibilities.

Using our heuristic, we were able to find a set of approximated Pareto-optimal solutions to this topology optimization problem. Whereas, the hybrid ILP/EA-based approach having failed completely caused by the huge ILP models. Thus, we are not able to present quantitative results for this case study. However, this example shows an application area where our approach for topology optimization helps a system-level designer for an unbiased decision making.

V. Conclusion and Future Work

In this paper, we presented a framework topology optimization for networked embedded systems. The input specification is given by a specification graph permitting modeling of demands for heterogeneous networks. The novelties may be summarized as follows: a) Our specification graph enables the modeling of routing restrictions, for example if a certain type of demand cannot be routed over some parts in a network. In addition to the model, b) we proposed a new chromosome encoding and a novel heuristic for demand binding. c) The performance of the proposed strategy has been compared with an ILP-based approach and it has been shown that it performs very well. Moreover, the applicability to recent design issues in the field of automotive networks has been presented. All in all, the presented framework provides a first methodology for multi-objective exploration of heterogeneous networked

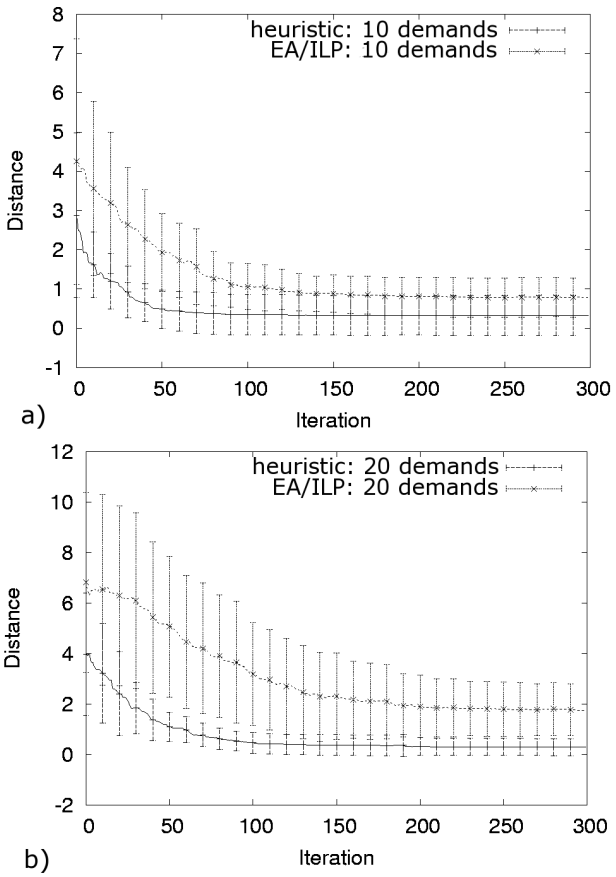

Fig. 4. Comparison of heuristic and hybrid ILP/EA optimization. The distance is presented for a) 10 different modeled demands and b) 20 demands over the iterations. dashed line: distance of the hybrid ILP/EA based optimization method, continuous line: distance of the proposed method between the iteration runs and a reference set.

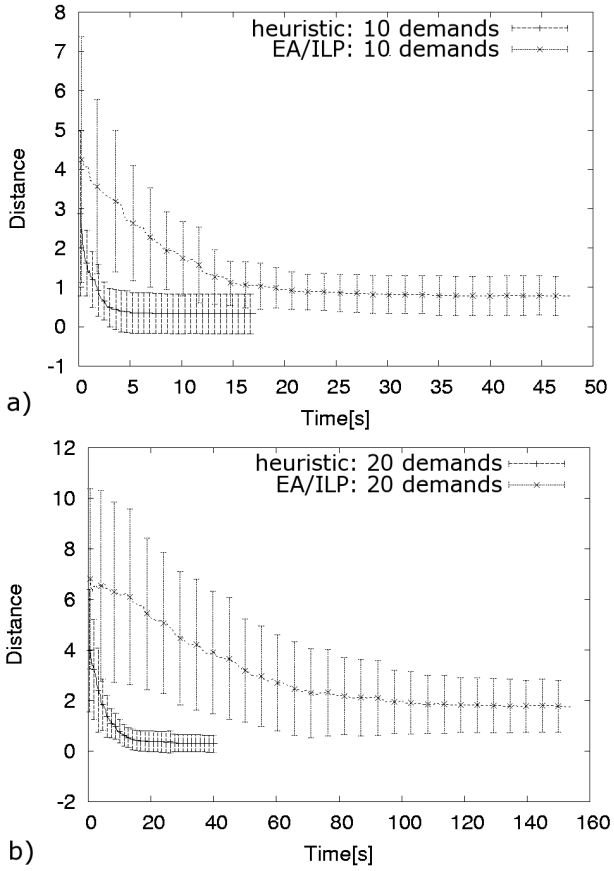

Fig. 5. Comparison of heuristic and hybrid ILP/EA optimization. The distance is presented for a) 10 different modeled demands and b) 20 demands over the exploration time. dashed line: distance of the hybrid ILP/EA based optimization method, continuous line: distance of the proposed method between the iteration runs and a reference set.

embedded systems.

REFERENCES

[1] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli, "Metropolis: An Integrated Electronic System Design Environment," *IEEE Computer*, vol. 36, no. 4, pp. 45–52, Apr. 2003.

[2] S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler, "PISA - A Platform and Programming Language Independent Interface for Search Algorithms," in *Proc. of the 2nd Int. Conf. on Evolutionary Multi-Criterion Optimization*, vol. 2632, Faro, Protugal, Apr. 2003, pp. 494–508.

[3] T. Blickle, J. Teich, and L. Thiele, "System-Level Synthesis Using Evolutionary Algorithms," in *Design Automation for Embedded Systems*, ser. 3, R. Gupta, Ed. Boston: Kluwer Academic Publishers, Jan. 1998, pp. 23–62.

[4] CAN, "Controller Area Network," http://www.can.bosch.com/.

[5] FlexRay, http://www.flexray.com/.

[6] N. Garg and J. Könemann, "Faster and Simpler Algorithms for Multicommodity Flow and other Fractional Packing Problems," in *Proceedings of 39th Annual Symposium on Foundations of Computer Science, FOCS'98*, Palo Alto, California, USA, Nov. 1998.

[7] V. Kianzad and S. S. Bhattacharyya, "CHARMED: A Multi-Objective Co-Synthesis Framework for Multi-Mode Embedded Systems," in *Proc. of the 15th IEEE Int. Conf. on Application-Specific Systems, Architectures and Processors (ASAP'04)*, Galveston, U.S.A., Sept. 2004, pp. 28–40.

[8] M. Laumanns, "Analysis and Applications of Evolutionary Multiobjective Optimization Algorithms," Ph.D. dissertation, Eidgenössische Technische Hochschule Zürich, Aug. 2003.

[9] P. Lieverse, P. van der Wolf, and E. Deprettere, "A Trace Transformation Technique for Communication Refinement," in *Proceedings 9th International Symposium on Hardware/Software Codesign (CODES'2001)*, Copenhagen, Denmark, Apr. 2001, pp. 134–139.

[10] LIN-Subbus, "LocalInterconnect Network," http://www.lin-subbus.org/.

[11] LpSolve, *Reference Guide*, 2005, lpsolve.sourceforge.net.

[12] S. Mohanty, V. K. Prasanna, S. Neema, and J. Davis, "Rapid Design Space Exploration of Heterogeneous Embedded Systems Using Symbolic Search and Multi-Granular Simulation," in *Proc. of the joint conf. on Languages, compilers and tools for embedded systems: software and compilers for embedded systems*, Berlin, Germany, June 2002, pp. 18–27.

[13] A. D. Pimentel, L. O. Hertzberger, P. Lieverse, P. van der Wolf, and E. F. Deprettere, "Exploring Embedded-Systems Architectures with Artemis," *IEEE Computer*, pp. 57–63, Nov. 2001.

[14] L. Thiele, S. Chakraborty, M. Gries, and S. Künzli, "A Framework for Evaluating Design Tradeoffs in Packet Processing Architectures," in *Proc. of the 39th Design Automation Conference (DAC 2002)*, New Orleans LA, USA, June 2002, pp. 880–885.

[15] TTP, "Time Triggered Protocol," http://www.TTTech.com/.

[16] E. Zitzler, "Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications," Ph.D. dissertation, Eidgenössische Technische Hochschule Zürich, Nov. 1999.