Exploration of Distributed Automotive Systems using Compositional Timing Analysis

Martin Lukasiewycz, Michael Glaß, Jürgen Teich, and Samarjit Chakraborty

Abstract This paper presents a design space exploration method for mixed event-triggered and time-triggered real-time systems in the automotive domain. A design space exploration model is used that is capable of modeling and optimizing state-of-the-art automotive systems including the resource allocation, task distribution, message routing, and scheduling. The optimization is based on a heuristic approach that iteratively improves the system design. Within this iterative optimization it is necessary to analyze each system design where one of the major design objectives that needs to be evaluated is the timing behavior. Since timing analysis is a very complex design task with high computational demands, it might become a bottleneck within the design space exploration. As a remedy, a clustering strategy is presented that is capable of reducing the complexity and minimizing the runtime of the timing analysis. A case study gives evidence of the efficiency of the proposed approach.

1 Introduction

Automotive electronics are constantly becoming more complex due to the innovation pressure in the automotive domain. A vast majority of innovations in the au-

Martin Lukasiewycz
TUM CREATE, Singapore, e-mail: `martin.lukasiewycz@tum-create.edu.sg`

Michael Glaß
University of Erlangen-Nuremberg, Erlangen, Germany e-mail: `glass@cs.fau.de`

Jürgen Teich
University of Erlangen-Nuremberg, Erlangen, Germany e-mail: `teich@cs.fau.de`

Samarjit Chakraorty
TU Munich, Munich, Germany e-mail: `samarjit@tum.de`

This work was financially supported in part by the Singapore National Research Foundation under its Campus for Research Excellence And Technological Enterprise (CREATE) programme.

tomotive domain is nowadays driven by embedded systems. In the last years such innovations were for example adaptive cruise control, pedestrian detection, or intelligent parking assist systems. However, these innovations require increasingly sophisticated system architectures. As a result, top-of-the-range vehicles already contain up to 100 Electronic Control Units (ECUs) and a multitude of different bus systems. In case functions have stringent latency and jitter constraints, these systems often require a complex validation of end-to-end timing behavior, using tools like Symbolic Timing Analysis for Systems (SymTA/S) [12] or Modular Performance Analysis (MPA) [2, 4]. This evaluation is a challenging design task and might become a bottleneck within a Design Space Exploration (DSE) where an optimization of the resource allocation, task mapping, message routing, and scheduling is performed. As a remedy, this paper presents a DSE approach that uses efficient timing analysis based on a graph-based representation and a fine-grained fixed-point iteration that partitions the problem in case of cyclic dependencies. In the following, the DSE model is introduced. Based on this model, an approach is presented that is capable of reducing the runtime of the timing analysis significantly. This is done by a decomposition of the timing analysis problem and an ordered evaluation. Finally, a case study is presented that gives evidence of the efficiency of the proposed approach.

2 Design Space Exploration Model

In the following, the Design Space Exploration (DSE) model is introduced, see [3, 10]. It is based on the optimization approach presented in [9]. This optimization approach is based on an Evolutionary Algorithm (EA), supporting multiple and non-linear objectives. For this purpose it becomes necessary to define the model formally and encode it into a set of linear constraints with binary variables such that a feasible implementation corresponds to a feasible solution. The remaining optimization, including load balancing and non-linear constraint satisfaction, is automatically carried out in an iterative search process of the optimization approach. This procedure significantly reduces the efforts to implement a new optimization approach or define complex constraints to direct the search towards the optimal implementations.

2.1 Model Description

The used exploration model is defined by a *specification* that consists of an *application* and an *architecture*. *Mappings* represent the relation between the process tasks from the application to the architecture, indicating which process task can be implemented on which resource. The application model also supports multi-cast and multi-hop communication by introducing messages additionally to the process tasks. A message can be routed on every resource except those where a routing is explicitly prohibited. From this specification, various *implementations* are derived. The implementation is defined by the *allocation* of architecture resources from a set of predefined components, the *binding* of process tasks to resources and *routing* of messages. The Y-chart approach for this model is illustrated in Figure 1.

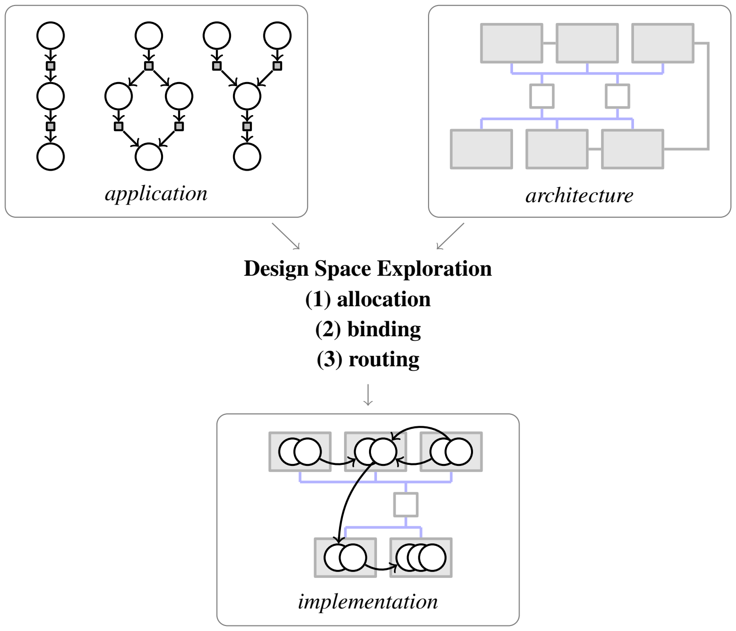

Fig. 1 Illustration of the Y-chart approach for the Design Space Exploration (DSE) model. An application is mapped to an architecture, resulting in an implementation. The DSE performs an allocation of resources, binding of process tasks, and routing of messages.

The specification consists of an architecture graph G_R, an application graph G_T, and mapping edges E_M:

- The architecture is given by a directed graph $G_R(R, E_R)$. The vertices R represent resources such as ECUs, gateways, and bus systems. The directed edges $E_R \subseteq R \times R$ indicate available communication connections between resources.
- The application is given by a directed graph $G_T(T, E_T)$ with $T = P \cup C$. The vertices T are either process tasks $p \in P$ or messages $c \in C$. Each edge $e \in E_T$ connects a vertex in P to one in C, or vice versa. Each process task can have multiple incoming edges that indicate the data dependencies to communication information of the predecessor messages. A process task can also have multiple outgoing edges to allow the sending of multiple different messages. On the other hand, each message has exactly one predecessor process task as the sender. To allow multi-cast communication, each message can have multiple successor process tasks.

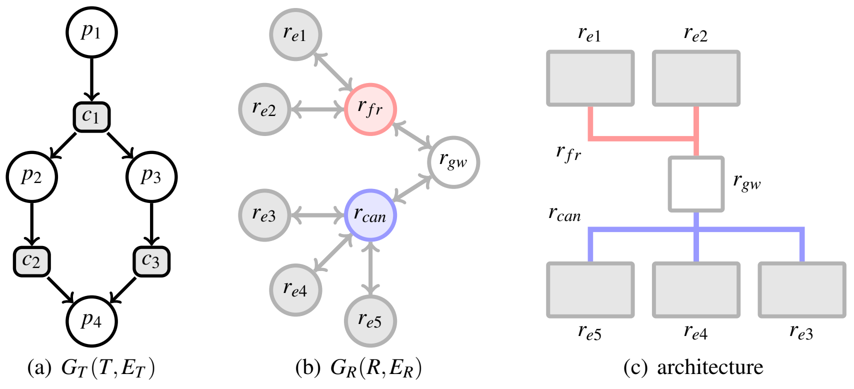

Fig. 2 Specification with the application graph G_T (a) and architecture graph G_R (b) for a given architecture (c). The mapping edges are defined as follows: $E_M = \{(p_1, r_{e1}), (p_1, r_{e2}), (p_2, r_{e2}), (p_3, r_{e2}), (p_3, r_{e3}), (p_3, r_{e4}), (p_4, r_{e5})\}$.

- The set of mapping edges E_M contains the mapping information for the process tasks. Each mapping edge $m = (p, r) \in E_M$ indicates a possible implementation of the process $p \in P$ on the resource $r \in R$. Without loss of generality it is assumed that messages can be routed on every resource.

A sample specification is given in Figure 2. This specification comprises a Control Area Network (CAN) bus (r_{can}), a FlexRay bus (r_{fr}), and a gateway (r_{gw}) that interconnects the buses. The communication over the buses and the gateway can only be established by multiple hops.

One implementation consists of the allocation graph G_α that is deduced from the architecture graph G_R and the binding E_β as a subset of E_M that maps the application to the allocation. Additionally, for each message $c \in C$ a sub-graph of the allocation $G_{\gamma,c}$ is determined that fulfills the data dependencies such that the communication is established between each sender process task and the corresponding receiver process tasks.

- The allocation is a directed graph $G_\alpha(\alpha, E_\alpha)$ that is an induced sub-graph of the architecture graph G_R. The allocation contains all resources that are available in the current implementation. The edges are induced from the graph G_R such that G_α is aware of all communication connections.
- The binding is performed by a mapping of the tasks to the allocated resources by deducing E_β from E_M such that the following requirements are fulfilled.
  Each process task $p \in P$ in the application is bound to exactly one resource:

$$\forall p \in P : |\{m | m = (p, r) \in E_\beta\}| = 1 \tag{1}$$

  Each task can only be bound to allocated resources:

$$\forall m = (p, r) \in E_\beta : r \in \alpha \tag{2}$$

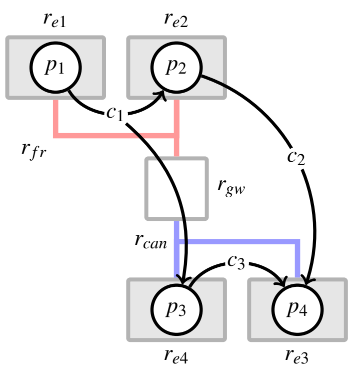

Fig. 3 Implementation for the specification in Figure 2. Illustrated is the allocation G_α, binding E_β, and routing G_γ. All routings are performed within multiple hops using the available buses and the gateway.

- Each message in $c \in C$ is routed on a tree $G_{\gamma,c}$ that is a sub-graph of the allocation G_α. The routings have to be performed such that all data dependencies given by the following two conditions are satisfied.
  For each message $c \in C$, the root of the routing has to equal the binding resource of the predecessor sender process task $p \in P$:

$$\forall (p,c) \in E_T, m = (p,r) \in E_\beta : |\{e | e = (\tilde{r}, r) \in G_{\gamma,c}\}| = 0 \tag{3}$$

  Each message $c \in C$ has to be routed on the same resource as the binding resource of the successive process tasks $p \in P$:

$$\forall (c,p) \in E_T, m = (p,r) \in E_\beta : r \in G_{\gamma,c} \tag{4}$$

An implementation is *feasible* if all requirements on the process task binding and the routing of the messages, i. e., the data dependencies, are fulfilled. A feasible implementation for the specification in Figure 2 is given in Figure 3.

2.2 *Binary Encoding*

In the following, a set of linear constraints with binary variables is defined such that a solution $\mathbf{x} \in \{0,1\}^n$ corresponds to a *feasible* implementation x for the given DSE problem. The symbolic encoding uses of the following binary variables:

r - binary variable for each resource $r \in R$ indicating whether this resource is in the allocation α (1) or not (0)
m - binary variable for each mapping $m \in E_M$ indicating whether the mapping edge is in E_β (1) or not (0)

$\mathbf{c_r}$ - binary variable for each message $c \in C$ and the available resources $r \in R$ indicating whether the message is routed on the resource (1) or not (0)
$\mathbf{c_{r,t}}$ - binary variable for each message $c \in C$ and resource $r \in R$ indicating on which communication step $t \in \mathscr{T} = \{1,..,|\mathscr{T}|\}$ (messages are propagated in steps or hops, respectively) a message is routed on the resource

The linear constraints are formulated as follows:
$\forall p \in P:$
$$\sum_{m=(p,r)\in E_M} \mathbf{m} = 1 \tag{5}$$
$\forall m = (p,r) \in E_M:$
$$\mathbf{r} - \mathbf{m} \geq 0 \tag{6}$$
$\forall c \in C, r \in R, (c,p) \in E_T, m = (p,r) \in E_M:$
$$\mathbf{c_r} - \mathbf{m} \geq 0 \tag{7}$$
$\forall c \in C:$
$$\sum_{r\in R} \mathbf{c_{r,1}} = 1 \tag{8}$$
$\forall c \in C, r \in R, (p,c) \in E_T, m = (p,r) \in E_M:$
$$\mathbf{m} - \mathbf{c_{r,1}} = 0 \tag{9}$$
$\forall c \in C, r \in R:$
$$\sum_{t\in\mathscr{T}} \mathbf{c_{r,t}} \leq 1 \tag{10}$$
$\forall c \in C, r \in R:$
$$\left(\sum_{t\in\mathscr{T}} \mathbf{c_{r,t}}\right) - \mathbf{c_r} \geq 0 \tag{11}$$
$\forall c \in C, r \in R, t \in \mathscr{T}:$
$$\mathbf{c_r} - \mathbf{c_{r,t}} \geq 0 \tag{12}$$
$\forall c \in C, r \in R, t = \{1,..,|\mathscr{T}|\}:$
$$\left(\sum_{\tilde{r}\in R, e=(\tilde{r},r)\in E_R} \mathbf{c_{\tilde{r},t}}\right) - \mathbf{c_{r,t+1}} \geq 0 \tag{13}$$
$\forall c \in C, r \in R:$
$$\mathbf{r} - \mathbf{c_r} \geq 0 \tag{14}$$
$\forall r \in R:$
$$\left(\sum_{c\in C \wedge r\in R} \mathbf{c_r}\right) + \left(\sum_{m=(p,r)\in E_M} \mathbf{m}\right) - \mathbf{r} \geq 0 \tag{15}$$

The constraints in Equation (5) and (6) fulfill the binding of each task to exactly one resource and the requirement that tasks are only bound to allocated resources, respectively, as stated in Equation (1) and (2). A message has to be routed on each target resource of the successive process task mapping targets as stated in the requirement in Equation (4). This requirement is fulfilled by the constraints in Equation (7). Analogously, as stated in the requirement in Equation (3), the constraints in Equation (8) and (9) imply that each message has exactly one root that equals the target resource of the predecessor mapping. The constraints in Equation (10) ensure that a message can pass a resource at most once such that cycles are prohibited. A message has to be existent in one communication step on a resource in order to be correctly routed on this resource as implied by the constraint in Equation (11) and (12). The constraints in Equation (13) state that a message may be routed only between adjacent resources in one communication step. In order ensure that the routing of each message is a sub-graph of the allocation, each message can be only routed on allocated resources as stated in the constraints in Equation (14). Additionally, the constraints in Equation (15) ensure that a resource is only allocated if it is used by at least one process or message such that suboptimal implementations are removed effectively from the search space. This minimizes the resulting allocation by redundant resources such that additional unnecessary costs are prohibited.

Given a single solution $\mathbf{x}$ of the defined set of linear constraints, a corresponding implementation x may be deduced as follows: The allocation G_α is deduced from the variables $\mathbf{r}$ and the binding E_β from the variables $\mathbf{m}$. For each message $c \in C$, the routing $G_{\gamma,c}$ is deduced from the variables $\mathbf{c_r}$ and $\mathbf{c_{r,t}}$.

3 Compositional Timing Analysis

The previous section defines a DSE model that is used to obtain feasible implementations x. Additionally, the exploration may also define priorities and schedules for the tasks and messages, respectively. For each implementation x, a timing analysis has to be performed to discard implementations that do not fulfill the real-time constraints of applications. In the following, a compositional timing analysis is proposed that is capable of determining end-to-end latencies efficiently in case of cyclic dependencies. Note that this approach reduces the runtime significantly without introducing any errors or additional over-approximations in the results.

3.1 Timing Model

In the used model it is assumed that the Worst-Case Execution Times (WCETs) of all tasks and the transmission times of messages are known. Also the periods of applications are predefined and the priorities of tasks and messages are either predefined or determined by the DSE.

The proposed compositional timing analysis approach may take advantage of different analysis techniques. For example, Modular Performance Analysis (MPA) [4] is used for modeling the FlexRay bus protocol. For analyzing the Control Area Network (CAN) bus, the approach presented by Tindell et al. [16] is applied. Here, it may be noted that these approaches have different mechanisms for representing timing properties of message streams. The approach in [16] uses the traditional period and jitter event model. On the other hand, MPA uses a more generic event model based on arrival curves. Further details on arrival curves may be found in [4]. A method to convert arrival curves into standard event models and vice versa is presented in [7]. Using this method, the proposed compositional timing analysis enables a hybrid approach such that standard event models like *periodic*, *periodic with jitter*, and *sporadic* might be used as well as arbitrary arrival patterns represented by appropriate arrival curves.

Though the timing analysis may be performed efficiently with the described models, cyclic dependencies in the timing analysis require a fixed-point iteration that may become computationally expensive. Due to the dependencies, the timing properties have to be calculated iteratively until there are no changes anymore. For this purpose, different approaches for an efficient fixed-point iteration for timing analysis are proposed. The model requires a graph-based representation of timing dependencies where the basic element is the *timing entity*. A timing entity might, for example, be the execution of a process on an ECU or a transmission of a message on a bus or gateway. The goal of the timing analysis is to determine the *timing properties* for each timing entity within a compositional approach, i. e., separately from other calculations. The timing properties are usually a delay and jitter where the jitter might be a single real value or an arrival curve known from MPA.

In the following, a common global dependency-based fixed-point iteration as well as the proposed fine-grained fixed-point iteration approach are presented. The automotive network in Figure 4 is introduced as an example to illustrate the proposed approaches.

3.2 Dependency-based Fixed-Point Iteration

The determination of the timing properties of a timing entity might depend on the timing properties of other entities. It is suggested to use a graph-based representation with $G_\chi(V_\chi, E_\chi)$ where V_χ is the set of timing entities and E_χ a set of directed edges that define the dependency between timing entities. An edge $(v, \tilde{v}) \in E_\chi$ indicates that the determination of the timing properties for $\tilde{v}$ depends on the timing properties of v. If such a *dependency graph* G_χ is acyclic, the timing properties may be determined in the partial order of the graph defined by the directed edges. In case of cycles in the graph, a *fixed-point iteration* becomes necessary to determine the timing properties of all entities. The dependency graph for the automotive network in Figure 4 is given in Figure 5 (a):

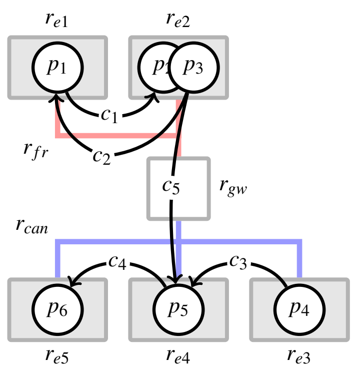

Fig. 4 A small automotive network consisting of five ECUs ($\{r_{e1},...,r_{e5}\}$), a CAN bus (r_{can}), a FlexRay bus (r_{fr}), and a gateway (r_{gw}). For the FlexRay bus, the static segment is used. The function consists of six processes ($\{p_1,...,p_6\}$) communicating via five messages ($\{c_1,...,c_5\}$). The index of the processes and messages represents the priority: A small number implies a high priority.

- All tasks or messages have an influence on the successive tasks or messages, respectively.
- Each process task on an ECU may delay the lower priority tasks on the same ECU (p_2 and p_3).
- Each message on the CAN bus may delay lower priority messages on the same CAN bus (c_3,c_4, and c_5).
- All messages on the FlexRay bus do not influence each other directly since they are routed on the static segment using Time Division Multiple Access (TDMA) (c_1 and c_2).

An algorithm for a dependency-based fixed-point iteration is given in Algorithm 1: The algorithm determines a fixed point for a subset $V \subseteq V_\chi$ of all timing

Algorithm 1 Dependency-based fixed-point iteration that is applied on a subset $V \subseteq V_\chi$ of timing entities.

```
1: $V_a = V$
2: while $V_a \neq \{\}$ do
3:     $v \in V_a$
4:     $V_a = V_a \setminus \{v\}$
5:     if $t_i(v) \neq t_{i-1}(v)$ then
6:         $V_a = V_a \cup (\{\tilde{v} | (v, \tilde{v}) \in E_\chi\} \cap V)$
7:     end if
8: end while
```

entities. The set V_a contains all timing entities that shall be evaluated, i. e., starting with V (line 1). The iterative algorithm proceeds until the set V_a is empty (line 2).

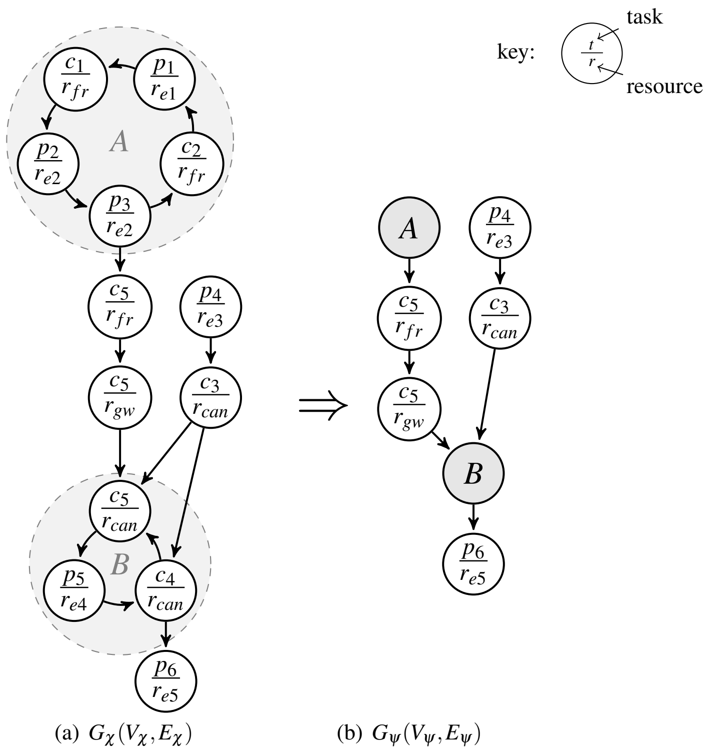

Fig. 5 Dependency graph G_χ (a) and acyclic dependency graph G_ψ (b) with merged states $A = \{(p_1, r_{e1}), (p_2, r_{e2}), (p_3, r_{e2}), (c_1, r_{fr}), (c_2, r_{fr})\}$ and $B = \{(p_5, r_{e4}), (c_4, r_{can}), (c_5, r_{can})\}$ for the automotive network in Figure 4.

In each iteration, one element from V_a is selected and removed (line 3,4). If the determined timing properties for v do not equal the previous value (line 5), all direct successive timing entities in G_χ that are also in V have to be re-evaluated (line 6). Note that the initial timing $t_0(v)$ for each entity, i. e., the initial jitter and delay, is 0 or is determined by initial timing approximations like presented in [13, 6].

Applying the Algorithm 1 to all timing entities, i. e., $V = V_\chi$, results a global dependency-based fixed-point iteration. Apparently, this approach is more efficient than a plain global fixed-point iteration that calculates the timing properties for all timing entities in each iteration until no value is changed. However, as shown in the following, this approach can be further improved by a fine-grained approach.

3.3 Fine-grained Fixed-Point Iteration

In the the following, a fixed-point iteration approach that is based on the stepwise calculation of timing entities is proposed. For the fine-grained fixed-point iteration, an acyclic graph $G_\psi(V_\psi, E_\psi)$ is deduced from G_χ. The vertices of the graph G_ψ are subsets of timing entities such that for each subset $V \in V_\psi$ it holds $V \subseteq V_\chi$. Furthermore, each timing entity is included in exactly one vertex:

$$\bigcup_{V \in V_\psi} V = V_\chi \tag{16}$$

$$\forall\, V, \tilde{V} \in V_\psi \text{ with } V \neq \tilde{V} : V \cap \tilde{V} = \{\} \tag{17}$$

Based on the partial order in G_ψ, a fixed-point iteration is applied to each node $V \in V_\psi$, i. e., the set of timing entities in V. These local fixed-point iterations are performed with the efficient dependency-based fixed-point iteration approach in Algorithm 1. Thus, a potentially inefficient global fixed-point iteration over all timing entities in V_χ is avoided. The acyclic dependency graph for the automotive network in Figure 4 is given in Figure 5 (b).

Given a dependency graph G_χ, an acyclic dependency graph G_ψ fulfilling the requirements in Equation (16) and (17) might be derived. To enable the best possible benefit from the introduced fine-grained fixed-point iteration, the optimal graph G_ψ shall contain a maximal number of vertices. A high number of vertices in G_ψ results in a high number of separate fine-grained fixed-point iteration steps and, thus, a more efficient approach is enabled.

In order to define the optimal graph G_ψ, the following reachability analysis is required. The function $r : V_\chi \to 2^{V_\chi}$ determines the set of reachable nodes in the graph G_χ from a node v and is defined recursively as follows:

$$r(v) = \{\tilde{v} \cup r(\tilde{v}) | (v, \tilde{v}) \in E_\chi\} \tag{18}$$

In a correct and optimal graph G_ψ, all timing entities with the same reachability are merged in the same vertex:

$$(\exists V \in V_\psi : v, \tilde{v} \in V) \Leftrightarrow r(v) = r(\tilde{v}) \tag{19}$$

The graph G_ψ is defined as *optimal* if only vertices with the same reachability are merged, i. e., the number of vertices in G_ψ is maximal. On the other hand, Equation (19) ensures that a graph G_ψ contains no cycles, i. e., it is *correct*.

An efficient approach that merges the vertices with the same reachability from G_χ to the vertices in G_ψ can be done in $O(n^2)$ with $n = |V_\psi|$ as presented in the following: Correspondingly to the forward-reachability from Equation (18), the backward-reachability is defined by $\bar{r} : V_\chi \to 2^{V_\chi}$ in the following recursive formulation:

$$\bar{r}(v) = \{\tilde{v} \cup \bar{r}(\tilde{v}) | (\tilde{v}, v) \in E_\chi\} \tag{20}$$

For any $v \in V_\chi$, the operation

$$V = r(v) \cap \bar{r}(v) \tag{21}$$

determines a vertex $V \in V_\psi$ containing all vertices on a cycle that contains v in G_χ. As a result, all vertices $\tilde{v} \in V$ have the same reachability $r(v)$ corresponding to Equation (19). Since Equation (21) corresponds to the definition of a strongly connected component, efficient algorithms from literature [1, 5, 14, 15] might be applied such that the complexity to remove all cycles becomes linear.

For small problems, this reduction of complexity may not be significant, in particular because the timing analysis is done once at design time. However, an efficient fixed-point iteration approach becomes highly important if one or more of the following attributes hold:

- Analysis of large real-world examples with hundreds of components resulting in a high number of timing entities.
- Detailed modeling of also different (software) layers resulting in a high number of timing entities.
- Timing analysis is applied within a DSE resulting in a high number of independent timing analysis calculations.

In this case, the presented approach significantly outperforms known global fixed-point iteration approaches. An evidence of the benefits of the fine-grained fixed-point iteration is given in the experimental results in the following section.

4 Experimental Results

In order to give evidence of the efficiency of the proposed approach, a case study is presented. All following experiments were carried out on an Intel Core 2 Quad 2.66 GHz machine with 3 GB RAM.

4.1 Automotive Case Study

We consider an automotive network exploration case study. The network architecture consists of 15 ECUs, connected via 2 CAN buses, 1 FlexRay bus, and a central gateway. The 9 sensors, and 5 actuators are connected via LIN buses to the ECUs. An application consisting of four functions, an *adaptive cruise control* (ACC), a *brake-by-wire* (BW), an *air conditioning function* (C1), and a *multimedia control* (C2), with 46 processes and 42 messages in total, is mapped to the given architecture. The functions and their real-time end-to-end constraints are listed in Table 1.

The functions are distributed according to state-of-the-art real-world networks where the ACC is implemented in the FlexRay sub-network, BW and C1 are implemented in one of the CAN sub-networks, and C2 is implemented over both CAN

function	#processes ($\|P\|$)	#messages ($\|C\|$)	max. latency [ms]
ACC	18	17	100
BW	8	7	50
C1	9	8	250
C2	10	9	150

Table 1 Detailed information about the functions of the used case study in terms of numbers of processes and messages as well as the maximal latency of each function.

sub-networks. This is regarded as the reference implementation. The reference implementation has the hardware cost of 216.80 € and an energy consumption of 11745 mA and fulfills all real-time constraints. Additionally, several mapping and resource alternatives are added to enable an effective DSE.

4.2 Design Space Exploration Results

To illustrate the advantages of the DSE and the presented timing analysis, the automotive network is optimized in terms of the hardware cost in Euro (€) and energy consumption in milliamperes (*mA*). The hardware costs are approximated by a linear function based on the cost per resource whiles additional costs for wiring, etc. are neglected. The energy consumption is approximated by a non-linear energy model based on the average utilization of the ECUs. The timing constraints are not linearizable and have to be handled by the EA separately such that implementations that do not fulfill these constraints are discarded. The DSE includes a concurrent optimization of parameters such as the priorities of the processes and messages as well as the scheduling of the messages on the static or dynamic segment of the FlexRay bus.

The optimization required 3511 seconds, using the fine-grained fixed point iteration. Within the optimization process, the EA obtained 5075 implementations that required a timing analysis. Note that an exploration with the dependency-based approach requires 3 hours and 120 seconds, leading to a significant speed-up already for the presented small case study. The plain global fixed point iteration requires more than one day (after which it was aborted). The results of the optimization are illustrated in Figure 6. Four non-dominated high quality implementations are found improving the reference implementation in both objectives, hardware cost and energy consumption. The found implementations decrease the hardware cost by 11.8% to 15% while the energy consumption is decreased by about 3.6% to 8.7% at the same time.

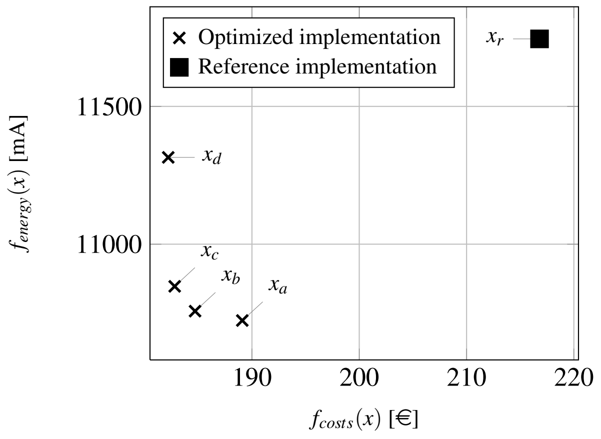

Fig. 6 The two dimensional plot of the optimization results and reference implementation of the automotive exploration case study.

4.3 Timing Analysis Results

In the following, we want to focus on the reference implementation and just vary the priorities and schedules to illustrate the strongly varying runtimes of the timing analysis approaches even for a small case study. Here, 100 different configurations are evaluated. A comparison to the plain global fixed point iteration is omitted due to the long runtimes of this method. The runtime of the global dependency-based approach is 192 seconds. The fine-grained approach including the generation of the acyclic dependency graphs requires only 70 seconds and, thus, improves the runtime of a single evaluation by a factor of approximately 2.75 on average. The runtimes for all 100 evaluations are illustrated in Figure 7. The plot shows that in many cases the runtime is approximately improved by a factor of two to four. On the other hand, the fine-grained fixed point iteration is also more than 8 times faster for some test cases even for this small case study.

The presented case study is rather small compared to real-world systems. For instance, in the automotive area, state-of-the-art architectures consist of up to 100 ECUs connected via several buses with hundreds of tasks and messages. Using the fine-grained fixed point iteration for such large systems shall improve the runtime of the timing analysis even more significantly. Moreover, the growing amount of computationally expensive timing analysis for some components that are based on Integer Linear Programming (ILP) [11] or model checking [8] require an efficient fixed point iteration approach in case of cyclic dependencies.

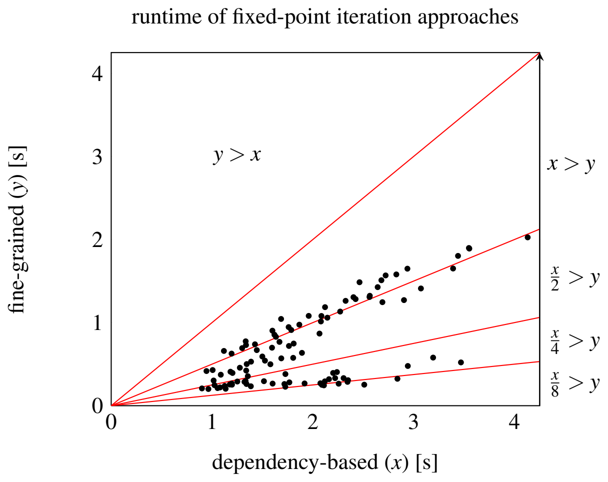

Fig. 7 Runtime comparison of different fixed point iteration approaches for 100 different priority-configurations for the reference implementation. Each dot specifies the runtime of the respective methods to determine the timing properties for a given implementation.

5 Concluding Remarks

This paper presents an efficient Design Space Exploration (DSE) using a fast timing analysis method. For the timing analysis, a timing entity graph is constructed and partitioned to achieve a fine-grained fixed point analysis. In future work, the proposed approach shall be combined with more complex timing analysis approaches that rely on ILP approaches or model checking. In this case, a fast timing analysis becomes inevitable if cyclic dependencies exist to significantly minimize the runtime of a DSE.

References

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: Data Structures and Algorithms (1983)
2. Anssi, S., Albers, K., Dörfel, M., Gérard, S.: ChronVAL/ChronSIM: A Tool Suite for Timing Analysis of Automotive Applications. In: Proceedings of the Conference on Embedded Real-time Software and Systems (ERTS 2012) (2012)
3. Blickle, T., Teich, J., Thiele, L.: System-Level Synthesis Using Evolutionary Algorithms. Design Automation for Embedded Systems **3**(1), 23–58 (1998)
4. Chakraborty, S., Kunzli, S., Thiele, L.: A General Framework for Analysing System Properties in Platform-based Embedded System Designs. In: Proceedings of the Conference on Design, Automation and Test in Europe (DATE 2003), pp. 190–195 (2003)
5. Cheriyan, J., Mehlhorn, K.: Algorithms for Dense Graphs and Networks on the Random Access Computer. Algorithmica **15**(6), 521–549 (1996)

6. Jonsson, B., Perathoner, S., Thiele, L., Yi, W.: Cyclic Dependencies in Modular Performance Analysis. In: Proceedings of the 8th ACM International Conference on Embedded software (EMSOFT 2008), pp. 179–188 (2008)
7. Künzli, S., Hamann, A., Ernst, R., Thiele, L.: Combined Approach to System Level Performance Analysis of Embedded Systems. In: Proceedings of the 5th IEEE/ACM International Conference on Hardware/software Codesign and System Synthesis (CODES+ISSS 2007), pp. 63–68 (2007)
8. Lampka, K., Perathoner, S., Thiele, L.: Analytic Real-time Analysis and Timed Automata: A Hybrid Method for Analyzing Embedded Real-time Systems. In: Proceedings of the 9th ACM International Conference on Embedded software (EMSOFT 2009), pp. 107–116 (2009)
9. Lukasiewycz, M., Glaß, M., Haubelt, C., Teich, J.: SAT-Decoding in Evolutionary Algorithms for Discrete Constrained Optimization Problems. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2007), pp. 935–942 (2007)
10. Lukasiewycz, M., Streubühr, M., Glaß, M., Haubelt, C., Teich, J.: Combined System Synthesis and Communication Architecture Exploration for MPSoCs. In: Proceedings of the Conference on Design, Automation and Test in Europe (DATE 2009), pp. 472–477 (2009)
11. Pop, T., Pop, P., Eles, P., Peng, Z., Andrei, A.: Timing Analysis of the FlexRay Communication Protocol. Real-Time Systems **39**(1), 205–235 (2008)
12. Richter, K., Ziegenbein, D., Jersak, M., Ernst, R.: Model Composition for Scheduling Analysis in Platform Design. In: Proceedings of the 39th Conference on Design Automation (DAC 2002), pp. 287–292 (2002)
13. Schioler, H., Jessen, J., Nielsen, J.D., Larsen, K.G.: Network Calculus for Real Time Analysis of Embedded Systems with Cyclic Task Dependencies. In: Proceedings of the 20th International Conference on Computers and Their Applications (CATA 2005), pp. 326–332 (2005)
14. Sedgewick, R.: Algorithms in C, Part 5: Graph Algorithms. Addison-Wesley (2002)
15. Tarjan, R.: Depth-first Search and Linear Graph Algorithms. SIAM Journal on Computing **1**(2), 146–160 (1972)
16. Tindell, K., Burns, A., Wellings, A.: Calculating Controller Area Network (CAN) Message Response Times. Control Engineering Practice **3**, 1163–1169 (1995)