Model-Based Energy Optimization of Automotive Control Systems

Joost-Pieter Katoen, Thomas Noll, Hao Wu†
Software Modeling and Verification Group
RWTH Aachen University, Germany
{katoen,noll,hao.wu}@cs.rwth-aachen.de

Thomas Santen, Dirk Seifert
Advanced Technology Labs (ATL) Europe
Microsoft Research, Aachen, Germany
{thomas.santen,dirk.seifert}@microsoft.com

***Abstract*—Reducing the energy consumption of controllers in vehicles requires sophisticated regulation mechanisms. Better power management can be enabled by allowing the controller to shut down sensors, actuators or embedded control units in a way that keeps the car safe and comfortable for the user, with the goal of optimizing the (average or maximal) energy consumption. This paper proposes an approach to systematically explore the design space of SW/HW mappings to determine energy-optimal deployments. It employs constraint-solving techniques for generating deployment candidates and probabilistic analyses for computing the expected energy consumption of the respective deployment. The feasibility and scalability of the method is demonstrated by several case studies.**

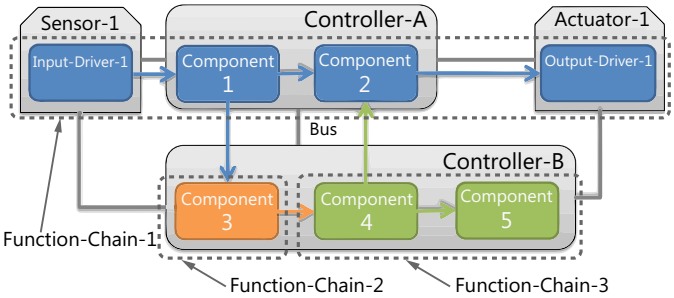

Fig. 1. An example deployment of function chains

I. Introduction

A. Problem Description

Nowadays, energy-saving activities in the transport domain mainly concentrates on the engines. But also the electronic control systems in vehicles significantly contribute to their energy consumption. Better power management can be enabled by allowing the controller to shut down sensors, actuators or embedded control units. As vehicles have to be considered as safety-critical systems, sophisticated regulation mechanisms are required to optimize the (average or maximal) energy consumption while guaranteeing the user's safety and comfort at all times. A key factor on which to base an optimization strategy is the operational mode of a vehicle, because not all functionality of a vehicle control system is used in all operational modes. Examples for operational modes are: parking, driving in the city, driving on a highway, and entertainment. Operational modes can be orthogonal to one another, i.e. several modes may be active simultaneously, such as parking and entertainment.

Software components do not work independently of one another. Moreover they form so called function chains, which operate together to perform a desired task. Among software components we distinguish between input/output drivers and functional components. In a deployment step the function chains must be mapped to existing hardware components. More precisely, input and output drivers are mapped to sensors and actuators, and functional components are mapped to Electronic Control Units (ECUs). Hardware and software components can be required to operate within multiple function chains. That means they can be a member of many function chains by consuming inputs or providing outputs for other components. Each of the different operation modes of a car implies that a certain number of function chains are (potentially) operational and that other function chains need not be available.

As a function chain generally consists of many software components, all participating components are active if the function chain is. A hardware component must be active if there is an active software component in any active function chain that is deployed to this hardware component. Shutting down an ECU thus disables all software components that are deployed to this ECU or that depend on the attached sensors and actuators.

In today's systems the power management is quite conservative in disabling functionality. In this paper we present an approach to reduce the energy consumption of an automotive control system by optimizing the deployment of functionality, and subsequently exploiting the power management opportunities that the chosen deployment provides. In particular, our approach analyzes the deployment problem based on operational modes and behavioral profiles, so that we can propose function chain deployments that allow for shutting down parts of the system while optimizing energy consumption over time (average) or to avoid peak energy consumption.

The scenario in Fig. 1 illustrates three function chains, which are deployed to two different ECUs. Function chains can include input/output drivers, such as `Output-Driver-1`. In a specific operational mode,

†Supported by ATL and RWTH Aachen University Seed Fund

978-3-9815370-0-0/DATE13/©2013 EDAA

`Function-Chain-1` implicitly triggers a dependency on `Function-Chain-2`, because the result of that chain is required as an input for `Component-2`. `Function-Chain-3` is not required to support the other chains and therefore can be put in a low-power state. An analysis of design alternatives can reveal that moving or duplicating the software component of `Function-Chain-2` to `Controller-A` will allow the system management to power down `Controller-B` completely in that specific operational mode.

B. Our Optimization Approach in a Nutshell

Due to the complexity of such intelligent control systems, computer support for their design is indispensable. Our approach relies on exploring the design space of embedded systems using the FORMULA[1] tool [1], [2]. The latter is based on model-based design and logic programming and allows for the logical specification of non-functional requirements for the components of the system. This includes e.g., schedulability or communication constraints. The FORMULA modeling tool can then be used to systematically analyze the interactions between and the impacts of these requirements on platform mappings and design spaces. This allows to automatically generate deployment candidates, i.e., mappings of software to hardware components that satisfy the given requirements.

The FORMULA approach, however, does not allow to assess more quantitative aspects of the system, like its energy consumption. The analysis of such properties additionally requires to take the timed and/or stochastic behavior of components and users into account. It is therefore necessary to extend the model-based design method towards a *generate-and-test* approach which comprises two steps: first, FORMULA generates a set of deployment candidates fulfilling the given constraints. Second, dedicated analysis methods are applied to evaluate the interesting quantitative properties of the respective deployment. The latter generally requires additional information that FORMULA does not deal with such as usage profiles of function chains. Given the random nature of such profiles in automotive embedded systems, a natural candidate for the analysis of quantitative aspects of the SW/HW mapping are probabilistic techniques.

Fig. 2 gives a more detailed overview of our approach. The computation of the expected energy consumption (lower box) is essentially based on two inputs. The first is the usage profile, specified as a discrete-time Markov chain (DTMC; upper left box) over mode configurations. The second is the set of deployment candidates (upper middle box) as provided by FORMULA. The outcome is the optimal deployment, in the sense that the expected energy consumption (for the given usage profile) is minimal. The procedure will be detailed further in this paper and will be applied to some (abstract) case studies so as to indicate its scalability and feasibility.

[1] http://research.microsoft.com/en-us/projects/formula/

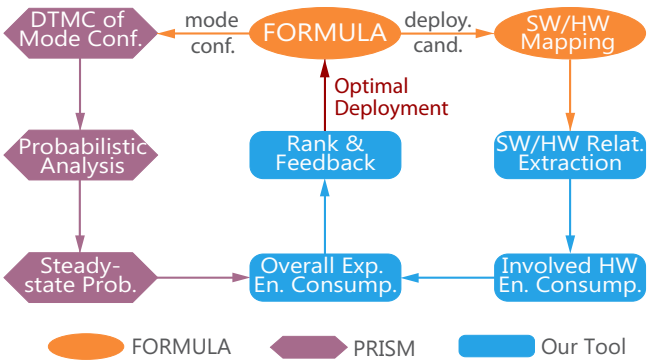

Fig. 2. Overview of our approach towards energy optimization of deployment candidates

C. Related Work

The main novelty of our approach is the combination of constraint-based generation of deployment candidates with the efficient quantitative analysis of Markov chains to analyse energy consumption. In our view, the constraint-based perspective is a natural fit to deployment generation, and conceptually differs from e.g., state-based techniques such as BIP as used in [3]. In particular, our approach allows to obtain useful information about energy optimization in an early design phase where detailed automata-based models are not yet at our disposal. Whereas most approaches are based on simulation [3] or analytical techniques [4], we advocate a numerical approach to determine the maximum expected overall energy consumption. Energy consumption analyses of single deployments include e.g., the analysis of selective deactivation of ECUs in networked embedded systems [5], and probabilistic model checking analysis [6]. Other approaches include e.g., the usage of non-linear convex programming techniques for determining optimal voltage supply [7].

II. The FORMULA Tool

Declarative specification languages with constraints are used in model-driven engineering to give formal semantics, define model transformations, and describe domain constraints. FORMULA (Formal Modeling Using Logic Programming and Analysis) [1], [2] is a modern formal specification language targeting model-based development (MBD). The core of FORMULA's approach uses *algebraic data types* (ADTs) and *strongly-typed constraint logic programming* (CLP), which support concise specifications of abstractions (domains) and model transformations. Specifications are understood as constraints on relations; the logic is a class of fixpoint logic (FPL), similar to (but more general than) Datalog.

In general, logic programs have a precise execution semantics, so domains can be understood programmatically. However, FORMULA's logic programs also have a precise interpretation as a system of first-order formulas, making them analyzable by existing tools. FORMULA applies the *satisfiability modulo theories* (SMT) solver Z3 [8] for search. A major advantage is its model finding and design space

exploration facility. It can be used to construct system models satisfying complex domain constraints. The user inputs a partially specified model, and FORMULA searches the space of completed models until it finds a globally satisfactory design. This process may be repeated to find many globally consistent designs. Variations on this procedure can be used to prove properties on model transformations and to perform bounded-symbolic model checking. Specifications are composable via a novel set of operators acting on data types and logic programs. They are constructive and they make semantic guarantees about composite domains.

III. The System Model

This section describes the system model in detail and explains the modeling of the usage profile.

A. System Structure

A key factor on which to base optimization strategies is the operational mode of a system, because not every functionality of a system is used in all modes. In turn, the functionalities or, more technically, function chains determine which (software and hardware) components need to be active. These entities are formally represented by finite sets of (operational) *modes*, Mod, *function chains*, Chn, and *software* and *hardware components*, SW and HW, respectively.

For the example system shown in Fig. 1 and using appropriate abbreviations, we obtain the following sets. (Note that the figure does not visualize the modes.)

- $Mod = \{\texttt{M1}, \texttt{M2}, \texttt{M3}\}$
- $Chn = \{\texttt{FC1}, \texttt{FC2}, \texttt{FC3}\}$
- $SW = \{\texttt{ID1}, \texttt{C1}, \ldots, \texttt{C5}, \texttt{OD1}\}$
- $HW = \{\texttt{S1}, \texttt{CA}, \texttt{CB}, \texttt{A1}\}$

Furthermore we need mappings that determine the relation between modes and function chains, between function chains and software components, and between software and hardware components. (In the following, $2^S := \{T \mid T \subseteq S\}$ denotes the *powerset* of a given set S.)

- $chn : Mod \to 2^{Chn}$: for every $m \in Mod$, $chn(m)$ yields the set of function chains that are *operative* in mode m;
- $sw : Chn \to 2^{SW}$: for every $c \in Chn$, $sw(c)$ yields the set of software components that are *engaged* in function chain c (where $sw(c_1)$ and $sw(c_2)$ are not necessarily disjoint for $c_1, c_2 \in Chn$); and
- $dpl : SW \to HW$ defines the *deployment* mapping, which determines for every $s \in SW$ the hardware component $dpl(s)$ on which software component s is running. We let Dpl denote the set of all such deployment mappings.

Thus the structure of the example system is formalized by letting $chn(\texttt{M1}) = \{\texttt{FC1}, \texttt{FC2}\}$, $sw(\texttt{FC1}) = \{\texttt{ID1}, \texttt{C1}, \texttt{C2}, \texttt{OD1}\}$, $dpl(\texttt{ID1}) = \texttt{S1}$, etc.

B. Cost Parameters

In order to determine the (expected) energy consumption of the system for a given deployment, the following quantitative parameters are required.

- the *cycle time* of the system (in ms), $\delta \in \mathbb{R}$, which determines the period during which its mode configuration remains unchanged;
- for every $s \in SW$, $t_{wc}(s) \in \mathbb{R}$ gives the *worst-case execution time* (WCET; in ms) for a single run of software component s;
- for every $s \in SW$, $f(s) \in \mathbb{N}$ gives the execution *frequency* of software component s, that is, the number of executions per cycle;
- for every $h \in HW$, P_{down}/$P_{idle}(h)$/$P_{act}(h) \in \mathbb{R}$ respectively determines the *power consumption* of hardware component h in shutdown/idle/active state (in W). Here a hardware component is considered to be *active* when it is currently executing a software component, otherwise *idle*. We assume that $P_{down}(h) \leq P_{idle}(h) \leq P_{act}(h)$ for every $h \in HW$.

To enable proper scheduling of software components, we moreover assume that $f(s) \cdot t_{wc}(s) \leq \delta$ for every $s \in SW$.

C. Behavioral Profile

With regard to the analysis of energy consumption, it is important to observe that operational modes can be orthogonal to each other, i.e., that several modes may be simultaneously active. The overall activity of the system is therefore determined by a (non-empty) set $M \subseteq Mod$ of currently active modes, called a *mode configuration*. As mentioned earlier, this configuration remains constant during the execution of a single cycle. Between cycles, however, it may change. We assume that inter-cycle transitions can be represented by a stochastic model which, for a given current configuration, enumerates all possible successor configurations that can be taken for the next cycle, together with their associated probabilities. Clearly, the successor configurations may include the current one. Formally, this model is a *discrete-time Markov chain (DTMC)*, a triple (S, s_0, p) where S is a countable set of *states* with *initial* state $s_0 \in S$ and $p : S \times S \to [0, 1]$ is a *probability matrix* satisfying $\sum_{s' \in S} p(s, s') = 1$ for each $s \in S$. Thus we assume that the mode transition behavior of the given system can be represented by a DTMC over mode configurations, i.e., with (finite) state space $S = 2^{Mod} \setminus \{\emptyset\}$.

For example, the diagram shown in Fig. 3 visualizes a DTMC with configurations over modes from the set $Mod = \{\texttt{M1}, \texttt{M2}, \texttt{M3}\}$. A possible interpretation is that M1 and M2 respectively represent the parking and driving mode of the vehicle while M3 indicates the activity of the entertainment system. Thus M1 and M2 exclude each other (and therefore do not occur jointly in any reachable configuration) while M3 is orthogonal to the other two modes. The initial state, $\{\texttt{M1}\}$, is marked by an ingoing arrow.

IV. The Analysis

The goal of our analysis is to rank the deployment candidates as provided by the FORMULA tool with respect to their (expected) energy consumption. Clearly the analysis has to employ probabilistic methods as the underlying behavior model, DTMCs, is of a stochastic nature. More concretely, for

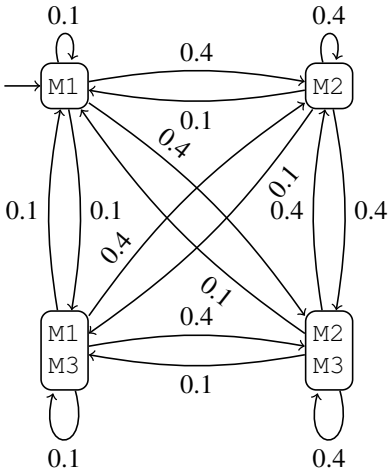

Fig. 3. A DTMC over mode configurations

a given deployment candidate, the expected energy consumption of the system using that deployment can be determined in the following way.

1) Compute the steady-state probabilities for all states in the DTMC.
2) Compute the energy consumption in each of the states, and attach it as reward information to the DTMC.
3) Compute the expected overall energy consumption per cycle as the expected reward.

These steps are detailed in the following.

A. Computation of Steady-State Probabilities

Evaluating the energy consumption of a system is based on computing the stationary distribution of the DTMC (S, s_0, p). It gives the probability of being in a certain state when taking a snapshot after a very long time or, in other words, the proportion of time that the DTMC "spends" in that state in the long run. Under certain mild assumptions regarding the DTMC (which usually apply in our setting), this distribution can be characterized by a flow balance equation system of the form

$$\pi = \pi \cdot p$$

which expresses that the total probability flow out of a state is equal to the total flow into the state.

The stationary distribution, that is, the solution π of the equation system can be computed using tools such as PRISM [9], based on the given DTMC. Note that this computation is independent of the concrete deployment. It therefore needs to be done only once in the beginning of the analysis before specific deployment candidates are considered.

B. Energy Consumption per Mode Configuration

Knowing the stationary distribution of the DTMC, the expected overall energy consumption per cycle with respect to a given deployment candidate can be determined by first computing the energy consumption per cycle for each of the mode configurations in the DTMC, and second weighting it with the stationary probability of the respective configuration. This section describes the first step in detail.

Given the current mode configuration $M \subseteq \mathit{Mod}$ (i.e., a state of the DTMC) and the deployment mapping $\mathit{dpl} \in \mathit{Dpl}$, the maximal energy consumption in mode M per cycle with respect to dpl can be computed as follows.

1) Determine the function chains that are operative in M:

$$\mathit{Chn}(M) := \bigcup_{m \in M} \mathit{chn}(m) \subseteq \mathit{Chn}.$$

2) Determine the engaged software components:

$$SW(M) := \bigcup_{c \in \mathit{Chn}(M)} \mathit{sw}(c) \subseteq SW.$$

3) Determine the engaged hardware components:

$$HW(M) := \mathit{dpl}(SW(M)) \subseteq HW.$$

4) For every $h \in HW(M)$:
   a) Determine the engaged software components that are deployed on h:

$$SW(h) := SW(M) \cap \mathit{dpl}^{-1}(h) \subseteq SW.$$

   Here $\mathit{dpl}^{-1} : HW \to 2^{SW}$ denotes the *inverse mapping* of dpl, defined by $\mathit{dpl}^{-1}(h) := \{s \in SW \mid \mathit{dpl}(s) = h\}$.
   b) For every $s \in SW(h)$, determine the active time of s (independent of mode M):

$$t_{act}(s) := f(s) \cdot t_{wc}(s).$$

   c) Determine the active time of h:

$$t_{act}(h) := \sum_{s \in SW(h)} t_{act}(s).$$

5) This yields the maximal energy consumption in mode configuration M per cycle:

$$\begin{aligned} E(M) &:= \textstyle\sum_{h \in HW \setminus HW(M)} P_{down}(h) \cdot \delta \\ &\quad + \textstyle\sum_{h \in HW(M)} E(h) \text{ where} \\ E(h) &:= P_{act}(h) \cdot t_{act}(h) + P_{idle}(h) \cdot (\delta - t_{act}(h)). \end{aligned}$$

Here, $E(h)$ is the expected energy usage of an engaged HW component expressed as weighted sum over the active time and idle time, respectively.

C. Expected Overall Energy Consumption

The expected overall energy consumption per cycle for the given deployment mapping can easily be obtained by computing the expected reward

$$E(\mathit{dpl}) = \sum_{M \subseteq \mathit{Mod}} \pi(M) \cdot E(M)$$

whose minimization yields the energy-optimal deployment.

#Mode conf.	#Modes	#FC	#SW	#HW	#Dpl.	Comp. time
4	3	3	7	4	16	1075 ms

TABLE I
OVERVIEW OF THE EXAMPLE

V. EVALUATION

The analysis described in the previous section has been implemented in a tool that, given a set of deployments determined by FORMULA, computes the expected energy consumption of each candidate. This section gives the result of some experiments to explore its capabilities. They were carried out on a machine equipped with an 3.06 GHz Intel Core i7 CPU and 5.8 GB of memory. We start with the evaluation of the running example, using 16 different deployments. Next we explore the scalability of our tool by changing certain model size parameters during deployment generation.

A. Running Example

In this experiment, we first use FORMULA to specify all parameters and constraints, and then generate a set of candidate deployments to compute their energy consumption. The parameters are those described in Section III, dealing both with the static part of the system structure (cf. Section III-A) and with its cost parameters (cf. Section III-B). Furthermore, FORMULA allows to specify some requirements that the generated deployments should satisfy, such as the schedulability of the resulting system, the minimal or maximal number of software components that are mapped to one hardware component, etc. Based on this information, FORMULA generates the possible deployments using SMT-based methods as sketched in Section II. We again refer to [1], [2] for details.

Each deployment is exported as a *4ml* file including additional information for energy consumption computation, and then our tool will import these files and extract the information for further processing. The steady-state probabilities of the mode configurations (cf. Section IV-A) are obtained by calling the PRISM[2] model-checking tool. Based on the results, our tool computes the expected energy consumption per cycle for each deployment and returns the deployment with minimal energy consumption.

The overview of the system parameters and the energy consumption distribution of the example are shown in Table I and Fig. 4, respectively. Furthermore, we set P_{down} of each hardware component be to 3% of its P_{act}. We observed that the major portion ($> 90\%$) of computation time is used to generate the PRISM representation of the DTMC model and to call PRISM for computing the steady-state probabilities for each operational scenario.

From the energy consumption distribution of 16 deployments, we can see that the average energy consumption is 430.8 mWs. The minimal energy consumption of 380.84 mWs can be achieved by choosing the 13th deployment, in which the software components `C1`, `C2` and `C3` are mapped to hardware

[2]http://www.prismmodelchecker.org/

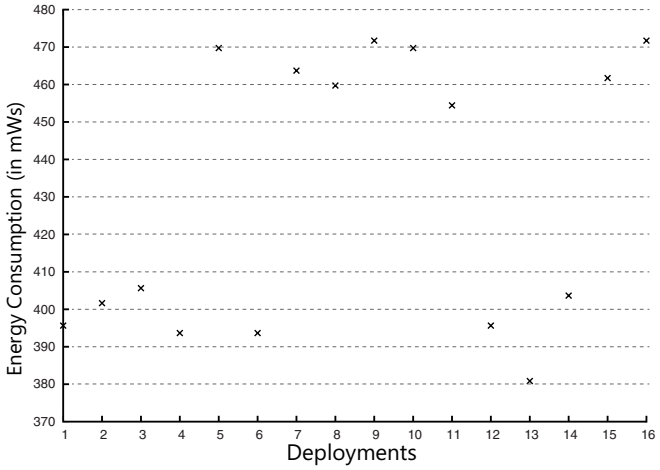

Fig. 4. Distribution of expected energy consumption per cycle

component `CB`, `C4` and `C5` are mapped to `CA`, `S1` is mapped to `ID1`, and `A1` is mapped to `OD1`. In contrast, both the 9th and 16th deployment have the maximal energy consumption of 471.72 mWs, which exceeds the minimal one by 90.88 mWs. From the results we observe that the deployments can be divided into two groups with 8 elements each: deployments 1, 2, 3, 4, 6, 12, 13 and 14 with relatively low and deployments 5, 7, 8, 9, 10, 11, 15 and 16 with relatively high energy consumption. This difference is caused by deploying `C3` to either `CB` or `CA`. There are three justifications why the first choice performs better: 1) `CA` has a much higher power consumption both in active and idle state than `CB`; 2) `C3` has the longest execution time during one cycle time among all software components; and 3) `C3` is involved in several mode configurations with a high probability of usage. Thus the optimal deployment can only be in first group. Then, the key factor of energy saving in these deployments is to fully use `CB`. Due to the restrictions, to fully use `CB` after `C3` is deployed to `CB`, there are three possibilities: 1) to deploy both `C1` and `C2` to `CB`, 2) to deploy `C4` to `CB`, 3) to deploy `C5` to `CB` (cf. deployments 13, 6 and 4 in Fig. 4, respectively). From Fig. 4 we can easily see that mapping `C1` and `C2` to `CB` yields the overall optimum.

B. Scalability Results

Next we tested the scalability of our tool by performing experiments on deployments with different parameter settings. We use another program to automatically create random deployments which satisfy the given parameters, such as the number of mode configurations, the number of function chains, etc. Also the number of generated deployments can be specified in this program. In our experiments, we compute the minimal energy consumption of 10 deployments generated by setting different parameters. Note that since we only specify the most important parameters, some parameters of the resulting deployments are randomly determined, such as the transition probabilities in the mode configuration DTMC. Table II shows the various parameters we use for testing and

#Mode conf.	#Modes	#FC	#SW	# HW
100-1000	100-1000	10-100	100-1000	100-1000

TABLE II
SCALABILITY PARAMETERS

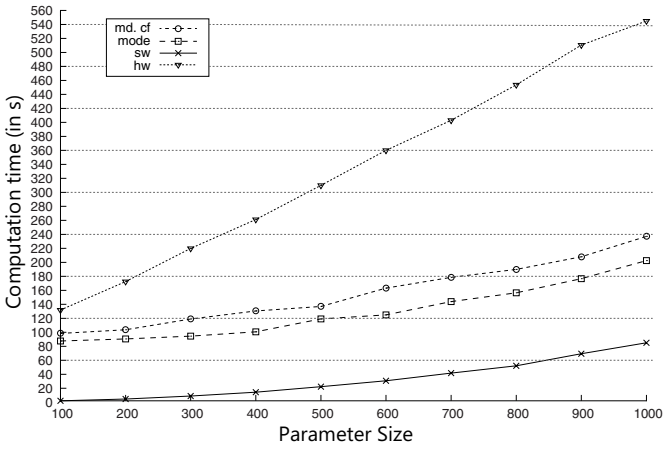

Fig. 5. Scalabilty test for configurations, modes, SW and HW components

their value ranges.

The computation time distributions for different parameter settings are shown in Fig. 5 and 6. The first successively varies the number of mode configurations (md. cf), modes, software components (sw) and hardware components (hw), which all increase from 100 to 1000 in steps of 100, respectively. Here, the sizes of the non-varying parameters (mode configurations, modes, software and hardware components) are fixed to 10, 5, 1000 and 10, respectively. Moreover the number of function chains of these generated deployments is fixed to 20. The results indicate that the number of hardware components has the strongest influence on the computation time whereas the other three parameters have a similar, weaker effect.

In addition, Fig. 6 shows that the number of function chains has an even more significant impact on the computation time. About 2122 s ($\approx$ 35 min) are needed to find the optimal deployment for 100 function chains whereas only 36 s are required for 10 chains. (The number of mode configurations, modes, software and hardware components is again fixed to 10/5/1000/10.) Finally we performed a maximal test (whose result is not shown in the figures) by computing 10 deployments for 100 mode configurations, 1000 modes, 100 function chains, 1000 software components and 100 hardware components. The computation takes about 5702 s ($\approx$ 95 min), which exceeds the longest time shown in Fig. 6 by a factor of about 2.7.

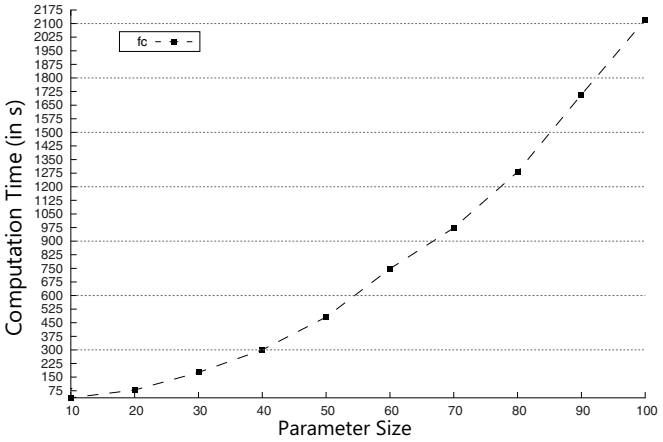

Fig. 6. Scalabilty test on function chains

VI. CONCLUSION

This paper presented a novel combination of constraint-based deployment generation and quantitative analysis to determine the SW/HW deployment that minimizes expected energy consumption. The approach has been implemented in a prototypical tool-chain using FORMULA and PRISM as tool components. Our experimental results show the sensitivity on several parameters and indicate a good scalability. Further research includes the application to automotive case studies.

REFERENCES

[1] E. K. Jackson, E. Kang, M. Dahlweid, D. Seifert, and T. Santen, "Components, platforms and possibilities: Towards generic automation for MDA," in *Proc. 10th Int. Conf. on Embedded Software (EMSOFT 10)*. ACM, 2010, pp. 39–48.

[2] E. K. Jackson, D. Seifert, M. Dahlweid, T. Santen, N. Bjørner, and W. Schulte, "Specifying and composing non-functional requirements in model-based development," in *Proc. 8th Int. Conf. on Software Composition (SC 2009)*, ser. LNCS, vol. 5634. Springer, 2009, pp. 72–89.

[3] P. Bourgos, A. Basu, M. Bozga, S. Bensalem, J. Sifakis, and K. Huang, "Rigorous system level modeling and analysis of mixed HW/SW systems," in *9th IEEE/ACM Int. Conf. on Formal Methods and Models for Codesign (MEMOCODE 2011)*. IEEE, 2011, pp. 11–20.

[4] B. Theelen, M. Geilen, T. Basten, J. Voeten, S. Gheorghita, and S. Stuijk, "A scenario-aware data flow model for combined long-run average and worst-case performance analysis," in *4th IEEE/ACM Int. Conf. on Formal Methods and Models for Codesign (MEMOCODE 2006)*. IEEE, 2006, pp. 185–194.

[5] C. Schmutzler, M. Simons, and J. Becker, "On demand dependent deactivation of automotive ECUs," in *Design, Automation & Test in Europe (DATE 2012)*. IEEE, 2012, pp. 69–74.

[6] G. Norman, D. Parker, M. Z. Kwiatkowska, S. K. Shukla, and R. Gupta, "Using probabilistic model checking for dynamic power management," *Formal Asp. Comput.*, vol. 17, no. 2, pp. 160–176, 2005.

[7] X. Piao, H. Kim, Y. Cho, M. Park, S. Han, M. Park, and S. Cho, "Energy consumption optimization of real-time embedded systems," in *Int. Conf. on Embedded Software and Systems (ICESS 2009)*. IEEE, 2009, pp. 281–287.

[8] L. M. D. Moura and N. Bjrner, *Z3: An Efficient SMT Solver*, 2008.

[9] M. Kwiatkowska, G. Norman, and D. Parker, "Probabilistic symbolic model checking with PRISM: a hybrid approach," *Int. J. on Software Tools for Technology Transfer*, vol. 6, no. 2, pp. 128–142, 2004.