Electronic Communications of the EASST

Volume MPM 2011 (
)

No \volumetitle defined!

Automatic Deployment Space Exploration Using Refinement Transformations

Joachim Denil, Antonio Cicchetti, Matthias Biehl, Paul De Meulenaere, Romina Eramo Serge Demeyer and Hans Vangheluwe

12 pages

No *ed(s) defined!
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Automatic Deployment Space Exploration Using Refinement Transformations

Joachim Denil[1,2], Antonio Cicchetti[3], Matthias Biehl[4], Paul De Meulenaere[2], Romina Eramo[5] Serge Demeyer[1] and Hans Vangheluwe[1,6]

[1] Joachim.Denil—Hans.Vangheluwe—Serge.Demeyer@ua.ac.be
Ansymo
University of Antwerp, Belgium
[2] Joachim.Denil—Paul.DeMeulenaere@kdg.be
TERA-Labs
Karel de Grote University College, Belgium
[3] antonio.cicchetti@mdh.se
School of Innovation, Design and Engineering
Mälardalen University (MDH), Västerås, Sweden
[4]biehl@md.kth.se
Embedded Control Systems
Royal Institute of Technology (KTH), Sweden
[5]romina.eramo@di.univaq.it
Computer Science Department
University of L'Aquila, L'Aquila, Italy
[6]hv@cs.ua.ac.be
Modelling, Simulation and Design Lab
McGill University, Montreal, Canada

Abstract: To manage the complex engineering information for real-time systems, the system under development can be described in a high-level architecture description language. This provides a basis for deployment space exploration. Subsequently the high-level information is used to create a low level implementation. However, in this mapping a lot of platform dependent choices have to be made and their consequences cannot be easily predicted. In this paper we present an approach to the automatic exploration of the deployment space based on platform-based design. All possible solutions of a deployment step are generated using a refinement transformation while the non-confirming results are pruned as early as possible using a simulation model or analytical method. We validate the feasibility of our approach by deploying part of an automotive power window optimized for its real-time behaviour using an AUTOSAR-like meta-model. First results are promising and show that the optimal solution can be found with our approach.

Keywords: MPM, MDE, Deployment space exploration

1 Introduction

Embedded systems often need to comply with particular requirements such as real-time execution deadlines, reliability and low power consumption. On the other hand, hardware and software platform resources are often limited, mainly due to cost reasons. An embedded systems platform is therefore often restricted in performance, memory, number of hardware interfaces, communication bandwidth, and so on. Building software applications on these restricted resources is not straightforward.

The software development process for these kinds of systems needs particular attention to deal with platform restrictions. During application design, the developer needs to make deployment choices in such a way that they match to the available platform resources, or even optimize their usage. Examples of these choices range from the mapping of software components to hardware platforms, though deployment decisions are typically based on engineering experience, on analysis, or on conducted experiments such as simulation or rapid prototyping. The optimization of the usage of platform resources will lead to an efficient use of the platform.

This paper will focus on the use of model transformations during the design process, with the goal of automatic deployment space exploration. During the deployment process, application models evolve from a high abstraction level down to a complete design. To use these application models for deployment optimization, it is often necessary to transform them into simulation or analytical models that allow estimation of the effect of certain resource limitations.

To illustrate the use of transformations in this process, we will follow the example of the power window. This application controls the up- and downward movements of a car window, and includes security issues such as the avoidance of any object being trapped by a closing window. The power window will be optimized for its real-time behaviour.

The rest of the paper is organised as follows: Section 2 gives a small introduction to the AUTOSAR platform and meta-model. Afterwards the work related to this work is stated in section 3. We elaborate our approach in section 4 and validate the feasibility of our method by deploying a part of the power window application in section 5. In section 6 we discuss our approach and identify the open issues. Finally section 7 concludes and states our future work.

2 Background

Dealing with the growing complexity of embedded systems demands for adequate support of development mechanisms; in fact, their usage often involves critical real-life aspects and a failure can cause catastrophic consequences. The automotive domain is not an exception, given the amount of electronically managed functionalities, ranging from infotainment to power window, cruise control, and braking system. As a consequence, several approaches emerged aiming at reducing problem intricacy by means of modelling and automation techniques, such as EAST-ADL [CFJ+08] and AUTOSAR [AUT08]. Nonetheless, the criticality and intrinsic complexity of automotive embedded systems make the development process still a time-consuming activity. Notably, in the deployment phase the developer has to provide platform details in terms of parameters, that will later influence the final product in terms of its properties, such as safety, performances[SD07]. We will use AUTOSAR as the framework for building our applications. AUTOSAR provides developers with a meta-model to develop applications and a middleware to

abstract the hardware.

The functional model of an AUTOSAR application consists of a set of *atomic software components*. These components can interact with each other using *ports*. The service or data provided or required by a port are defined by its *interface*. The interface defines the semantics of the transmission and can either be client-server or sender-receiver. Each software component defines its behaviour by means of a set of *runnables*. A runnable is a function that can be executed in response to *events*, for example a timing event. Figure 1 shows the application model of a power window.

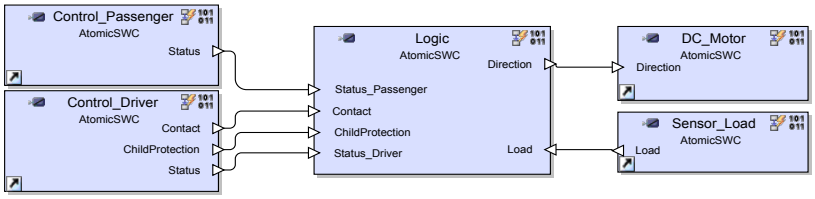

Figure 1: The software model of a power window application, showing the components and the interfaces for interaction

To make software components independent from the hardware, containing multiple Electronic Control Units (ECUs), the interface to this hardware must be standardized. This is done using the AUTOSAR basic software, shown in Figure 2.

This middleware consists of a real-time operating system based on the OSEK/VDX standard [OSE05]. The operating system schedules tasks in a fixed priority way. Since the concept of a task is not known at the functional level, the components must first be mapped to the processors and then the runnables must be mapped onto tasks. The mapping to tasks is not necessarily 1-to-1. The rules for mapping runnables to tasks are defined in the run-time environment (RTE) specification, available on [AUT11]. All tasks have to be assigned a priority to be scheduled by the operating system.

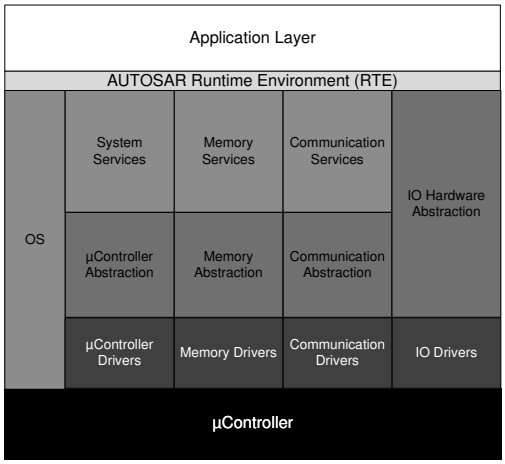

Figure 2: Structure of the AUTOSAR basic software, the run-time environment and the application layer

The middleware also contains services for sending and receiving messages on a communication bus. These are composed of signals that originate in the application layer. Communication signals and messages have certain configurable properties, such as the signal transfer property and the message trans-

mission mode, that have an impact on the timing behaviour of the application. More information about the configuration parameters in the communication stack can be found in the AUTOSAR communication specifications available on [AUT11].

On the communication abstraction and driver layer, the most common automotive buses, for example the Controller Area Network (CAN), are currently supported by the AUTOSAR communication stack. These also have many configuration parameters, such as the priority of the frames containing the message, that impact the real-time behaviour of the full system.

The RTE is used as a glue between the functional components and the AUTOSAR basic software. It is responsible for storing the internal messages using buffers or forwarding the external messages to the communication stack. It also activates the runnables when an event occurs.

It becomes evident that taking into account all the variability illustrated so far results in a huge amount of possible deployment combinations, which has to be typically managed by some domain expert. Such a task would greatly benefit of automation support, especially in determining available deployment alternatives at a given development stage.

3 Related work

Performance analysis is crucial for the deployment of safe and cost-effective software-intensive systems. In [BDIS04] a review of research in the field of model-based performance prediction is presented. The techniques are based on simulation models, process algebra, Petri nets and stochastic processes. An example of the discussed methods close to our research is architecture-based performance analysis [SG98] where a performance model, based on queuing networks, is derived from a system described in UML. For the design and deployment of software components Palladio [BKR09] offers a meta-model with annotations for extra-functional properties. The model can be transformed into both an analytical and a simulation model. Kugele et al. uses a similar approach by annotating a component-based meta-model with extra-functional properties, though part of the deployment is automated using an integer linear programming approach.

In the last decade platform-based design [KNRS00, SM01] received a lot of attention. Platform-based design introduces clear abstraction levels and allows for separation of concerns between the refinement of the functional architecture specification and the abstractions of possible implementations. Di Natale and Sangiovanni-Vincentelli adapted the platform-based technique in [SD07] to the development of automotive embedded systems. The definitions of the models and architecture solutions, involved in the AUTOSAR process, are isolated from the details while still allowing enough information for the accurate prediction of the implementation's properties. The process is driven by a what-if analysis.

Popovici et al. developed an exploration technique based on platform-based design for the deployment of multimedia applications on MPSoC architectures in [PGR+08]. The technique allows software code generation, software development platform generation and simulation model generation. This allows easy experimentation with different mappings of the software on the architecture. Different levels of abstraction are defined where the generation, simulation and validation of the software components can take place.

Another approach, DECOS [OPH06, HSS+07] (dependable embedded components and systems), uses model-based development techniques to develop complex distributed embedded systems. The DECOS architecture enables the transition from a federated to a more integrated

distributed architecture, integrating multiple subsystems onto a single platform. Because of this, the platform choices are less open and depend on a time-triggered communication bus. The developer is assisted during the deployment by a commercial tool, TTTech toolsuite, for schedulability analysis.

Besides these performance analysis methods and deployment space exploration techniques there are some other automatic methods for local optimization. These methods optimize a certain part of the deployment space. Some use heuristic search methods, like simulated annealing [PEP02], genetic algorithms [Sin07] or use a linear programming approach [ZZDS07] or SAT-based approach [MH06].

Schatz et al. developed a rule-based transformation technique [SHL10] based on Prolog to generate the full deployment space of an embedded system.

Finally, The DESERT tool-suite [NSKB03] provides a framework for design space exploration. It allows an automated search for designs that meet structural requirements. Possible solutions are coded in a binary encoding that can generate all possibilities. A pruning tool is used to allow the user to select the designs that meet the requirements. These can then be reconstructed by decoding the selected design.

4 Incremental refinement of deployment decisions

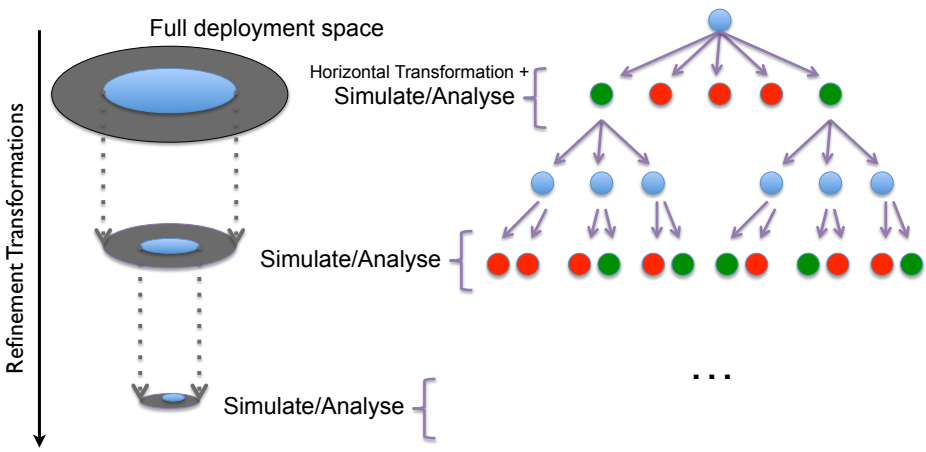

Figure 3: Approach

Our approach, shown in Figure 3, is based on the key concepts of multi-paradigm modelling: Model everything at the right level of abstraction using the correct formalism [VD02]. For this purpose two types of transformations are used:

- Refinement Transformations: Deployment can be viewed as a set of transformations since we iteratively add knowledge about the platform into the model. By using these transformations we could generate all possible solutions allowed within the meta-model. Since every transformation will yield multiple solutions, it will cause an explosion of the search

space. Therefore, it is important that non-conforming solutions are pruned as early as possible. To avoid this problem, multiple layers of abstraction are defined where high-level estimators can be found to check the generated partial solutions.

- Horizontal Transformations: The model is transformed to another formalism on the same abstraction level and evaluated using a simulation model or an analytical method.

The solution space is limited with every abstraction layer since the pruned branches are not further explored. The choice of analysis method needs to match the system properties that are optimized. Here, we are interested in the performance of the system. For the deployment exploration in the context of real-time behaviour we identified three abstraction levels. The defined levels are pragmatically chosen so they correspond to the current analytical and simulation methods available in the automotive industry. For each of these levels we describe what information has to be available and what method is used to acquire the high-level estimator.

The first defined level abstracts the system architecture. Here, the software components of the application are mapped to a specific hardware component. In case of multiple components, a bus connects these components. Since all information about timing of the triggers and execution time is known at this point, we can use a simple bin packing check to ensure that no single component or bus is overused at this level of abstraction.

The next abstraction level concentrates on the services provided by the communication stack and operating system. The runnables are grouped into tasks and they are given a priority. The same is done for the signals that are to be transmitted on the bus. These are grouped into messages and given a priority for the arbitration on the bus. Furthermore, signals and messages are given their transmission mode and property. At this abstraction level all information for schedulability analysis is present. This means that solutions that cannot make their end-to-end deadlines are automatically pruned.

Finally, the application is fully mapped to the hardware platform by defining hardware buffers for the reception and transmission of messages. The drivers and interfaces of the communication stack are configured and software buffers are defined. Some hardware-specific options are also configured. The solutions are checked for their real-time behaviour under the influence of buffering. We use a DEVS simulation model presented in [DVR+11] for this purpose. Solutions that have lost messages and/or that cannot make their end-to-end deadlines under the influence of buffering are pruned in this step.

4.1 Refinement transformations

The exploration of the solution space is made by means of the Janus Transformation Language (JTL) [CDEP11], which is a bidirectional model transformation language based on Answer-Set Programming (ASP) [GL88]. The transformation engine is based on a generation mechanism that first expands the set of possible solutions based on mapping rules and then refines such a set by applying constraints on the derived target models, such as metamodel conformance rules and additional desired characteristics[1]. In our case, solution space exploration can be reduced to an endogenous model transformation, where the source and target models both conform to the same metamodel. Then, starting from a certain source model containing an incomplete deployment at a

[1] JTL has a number of interesting features that go far beyond the scope of this paper. The interested reader can start from [CDEP11] to have an overview of its main characteristics and to understand how the transformation looks like.

given development stage, it is possible to derive target models representing available deployment alternatives for the next abstraction level. For instance, it is possible to specify that if a boolean flag is present in the source model, then two different target models will be generated, one for the flag set to true and one for flag set to false. Similarly, it is possible to generate multiple assignment possibilities of hardware buffers to handlers.

4.2 Horizontal transformations and pruning

It is our goal to pick the deployment configuration that promises desirable extra-functional behaviour. For each possible configuration that was calculated in the previous step, we predict and analyse the behaviour of the system. For each system configuration we build a custom simulator, by automatically transforming the configuration into an executable simulation or analysable model.

5 Case study

To show that the technique and technology is feasible, we apply it to part of the deployment of an automotive power window. In our case study, the first two abstraction levels in the process are finished and we start with models abstracted at the service level.

5.1 The power window

The software model of our power window case study is based on [PM04]. The application controls the window on the passenger side, though both passenger and driver are allowed to open or close the window. When an object is present while closing the window, it will automatically detect this and lower the window.

Figure 1 shows the application model of the power window application. The functional model of an AUTOSAR based power window consists of 5 atomic software components. All components contain a single runnable except for the Driver component containing 3 runnables, each for reading a single sensor. The runnables are triggered using a timing event every millisecond. Table 1 shows the execution timings of the different components on a 32-bit platform.

Runnables and states in runnables	
Control_Driver	8.96 μs
Control_Passenger	5.01 μs
Sensor_Load	81.2 μs
Logic	39.7 μs
DC_Motor	2.0 μs

Table 1: Execution times of the power window components

For the case study we implemented an AUTOSAR like meta-model with annotations for execution timing using the EMF Ecore framework [SBPM08]. We generate all possible solutions

starting from a single model that has been checked by schedulability analysis. The model's variation points contain:

- Transmit and Receive buffers: The number of hardware buffers in the CAN controller are bounded. Therefore they must be distributed between transmission and reception buffers. Once they are partitioned, the frames need to be assigned to a hardware buffer. Hardware buffers can be overwritten if a message has not been transmitted yet. This makes this a crucial parameter in the real-time behaviour of the whole system.
- CAN interface module software buffering: This flag enables or disables the use of software buffers in the CAN interface basic software module. If software buffering is allowed, the message can be stored in software when the hardware buffer is full.
- CAN Transmit Cancellation: The transmit cancellation flag enables the cancellation of a message inside the CAN hardware buffer when a higher priority message is available. This could impact the real-time behaviour of the system due to message that are transmitted in a different order.
- CAN Multiplexing: The CAN hardware normally checks the buffers in a linear fashion, transmitting the first buffer that is not empty. By allowing the multiplexing option, the buffers of CAN become a priority queue. It will always transmit the highest priority message first.

5.2 Implementation of the refinement operations

Based on the variabilities described above, a first transformation has been written that explores the solution space without any additional constraints. Given the number of variables and the corresponding possible evaluations, the execution yielded 192 solutions. However, as previously discussed it is possible to add some domain knowledge in the form of new constraints to the JTL transformation to narrow down the solution space. For instance, ECUs not transmitting any frames make the CAN cancellation, multiplexing, and software buffering flags obsolete. Therefore, the transformation specification has been enriched by adding such domain knowledge. Because the source model contained an ECU transmitting no frames (i.e., Passenger), the solution space has been cut down to 24 possible solutions. In Figure 4, an excerpt of the deployment alternatives generated by the JTL transformation is illustrated[2]. In particular, it is possible to see a generated configuration as conforming to the abstract syntax exploited by JTL.

5.3 Implementation of the horizontal transformations

The solutions generated from the refinement transformation are transformed into a DEVS simulation model, as described in [DVR+11]. This model is a parametrized simulation model written in Python. All control units, tasks, messages, buffers and their properties need to be specified in Python and executed by the pyDEVS simulation engine.

We use the MOF Model to Text Language (MTL) [Omg08] defined by the OMG to transform the configuration model into python source code for the DEVS simulation. MTL is a template-

[2] The details of the JTL transformation for alternatives exploration can not be put in this paper due to space restrictions. The interested reader can access the implementation at http://itl.di.univaq.it/downloads/JTLExplorer.rar.

```
%%%% Target Model 1
model(m1, x_smallAROut).
node(x_smallAROut,"x_//@theSystem/@ecu.0", x_Ecu).
node(x_smallAROut,"x_//@theSystem/@ecu.1", x_Ecu).
node(x_smallAROut,"x_//@theSystem/@ecu.0/@canifConfig",x_CanifConfig).
node(x_smallAROut,"x_//@theSystem/@ecu.0/@canifConfig",x_CanifConfig).
node(x_smallAROut,"x_//@theSystem/@ecu.1/@canifConfig",x_CanifConfig).
node(x_smallAROut,"x_//@theSystem/@ecu.0/@canConfig",x_CanConfig).
node(x_smallAROut,"x_//@theSystem/@ecu.1/@canConfig",x_CanConfig).
node(x_smallAROut,"x_//@theSystem/@ecu.0/@canifConfig/@ipduToHohMap.0", x_IpduToHohMap).
node(x_smallAROut,"x_//@theSystem/@ecu.0/@canifConfig/@ipduToHohMap.1", x_IpduToHohMap).
node(x_smallAROut,"x_//@theSystem/@ecu.1/@canifConfig/@ipduToHohMap.0", x_IpduToHohMap).
node(x_smallAROut,"x_//@theSystem/@ecu.1/@canifConfig/@ipduToHohMap.1", x_IpduToHohMap).
node(x_smallAROut,"x_//@theSystem/@ecu.0/@comConfig/@ipdus.0", x_TxIPDU).
node(x_smallAROut,"x_//@theSystem/@ecu.0/@comConfig/@ipdus.1", x_TxIPDU).
node(x_smallAROut,"x_//@theSystem/@ecu.1/@comConfig/@ipdus.0", x_RxIPDU).
node(x_smallAROut,"x_//@theSystem/@ecu.1/@comConfig/@ipdus.1", x_RxIPDU).
...
prop(x_smallAROut,"x_//@theSystem/@ecu.01", "x_//@theSystem/@ecu.0", x_name, x_Driver).
prop(x_smallAROut,"x_//@theSystem/@ecu.11", "x_//@theSystem/@ecu.1", x_name, x_Passenger).
prop(x_smallAROut,"x_//@theSystem/@ecu.0/@canConfig.0","x_//@theSystem/@ecu.0/@canConfig",x_cancellation,x_false).
prop(x_smallAROut,"x_//@theSystem/@ecu.0/@canConfig.1","x_//@theSystem/@ecu.0/@canConfig",x_multiplexing,x_false).
prop(x_smallAROut,"x_//@theSystem/@ecu.1/@canConfig.0","x_//@theSystem/@ecu.1/@canConfig",x_cancellation,x_false).
prop(x_smallAROut,"x_//@theSystem/@ecu.1/@canConfig.1","x_//@theSystem/@ecu.1/@canConfig",x_multiplexing,x_false).
prop(x_smallAROut,"x_//@theSystem/@ecu.0/@canifConfig.0","x_//@theSystem/@ecu.0/@canifConfig",x_enableTxBuffering,x_false).
prop(x_smallAROut,"x_//@theSystem/@ecu.1/@canifConfig.0","x_//@theSystem/@ecu.1/@canifConfig",x_enableTxBuffering,x_false).
...
```

Figure 4: An excerpt of deployment alternatives generated by the JTL transformation.

based transformation language, allowing us to output Python code and inject the configuration parameters from the model. The result of this transformation is customized python code for the DEVS simulation of the chosen configuration.

5.4 Results

The simulation calculated a metric based on the end-to-end latencies of the application, the response times of the tasks and the idle-time of the processors. If messages were lost, the score was reduced to zero.

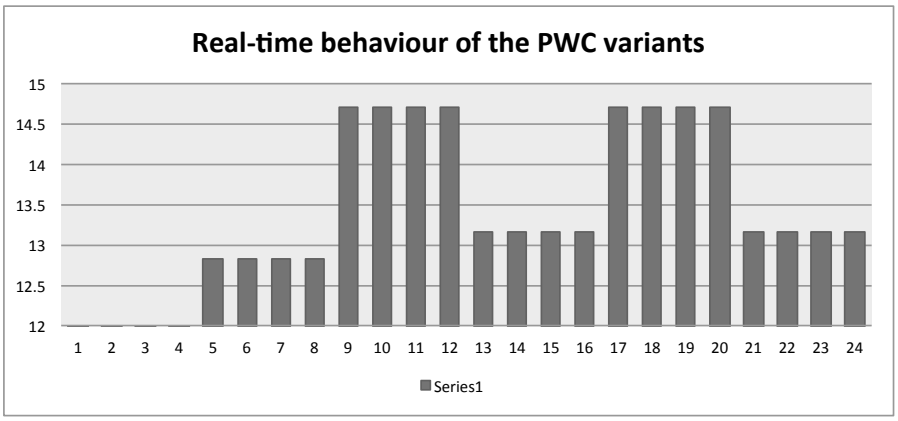

Figure 5: Real-time behaviour of the powerwindow case study (higher score is better)

Figure 5 shows the results of the 24 variants. Our method found 8 optimal solutions while four

solutions did not make the deadline since there were messages lost during execution.

The scores of the variants are very close to each other, which is due to the small case study. Since there are only 2 messages being transmitted, the effects of the multiplexing and cancellation is minimal since no messages need cancelling or multiplexing.

6 Lesson learned and open issues

One general issue of exploration techniques is the explosion of the search space. Technically, we can approach these problems by parallel computation, as the branches are independent. Each alternative can be simulated and refined independently of the other alternatives. Today, the AUTOSAR developer needs to explore the whole solution space manually, or apply best practices. Manual exploration is only possible for extremely small configuration spaces and best practices might miss the optimal combinations of deployment choices. On the other hand, grouping the deployment choices in different abstraction layers allows domain experts to focus on concepts closer to their expertise.

Our current approach targets the optimization of one system property, e.g., performance. The method could allow multi-objective optimization, allowing us to optimize for safety performance and cost. It will be necessary to combine several parameter evaluations at the same level of abstraction, entailing trade-off computations. The developers must then choose between all pareto-optimal solutions.

There is also a large amount of optimizations that can be added to this approach. If at a given point the solution fails, that branch is not explored deeper. There could be conditions by which further pruning of the solution space can be applied tracking the set of choices back in the tree branches. Other optimizations could be added by domain experts, as we did. They can define extra constraints to the exploration of the branches resulting in less solutions.

7 Conclusions and future directions

This paper proposed an automatic deployment exploration technique based on refinement transformations and platform-based design. We have shown that the techniques and technologies involved in the method are feasible. To illustrate the automatic exploration, we applied it to part of an automotive power window. The solution is optimized for its real-time behaviour using an AUTOSAR-like meta-model. First results are promising and show that the optimal solution can be found with our approach.

We will continue this work in four branches: (a) We will extend the transformations to include the 3 defined levels so a complete optimization of the power window can occur. (b) Other trade-offs can be included in the process, for example memory and safety requirements. (c) The power window is too small to verify whether the method is feasible for a real subsystem in the automotive domain. Therefore, bigger case studies will be done to investigate this. (d) By doing empirical research with this method, more domain knowledge could be discovered and included in the method so more non-optimal solutions can be pruned earlier on.

Bibliography

[AUT08] G. AUTOSAR. AUTOSAR methodology, 1.2.1, 2008. online: www.autosar.org.

[AUT11] AUTOSAR. AUTOSAR website. 2011.

[BDIS04] S. Balsamo, A. Di Marco, P. Inverardi, M. Simeoni. Model-based performance prediction in software development: a survey. *IEEE Transactions on Software Engineering* 30(5):295–310, May 2004.

[BKR09] S. Becker, H. Koziolek, R. Reussner. The Palladio component model for model-driven performance prediction. *Journal of Systems and Software* 82(1):3–22, 2009.

[CDEP11] A. Cicchetti, D. Di Ruscio, R. Eramo, A. Pierantonio. JTL: A Bidirectional and Change Propagating Transformation Language. In Malloy et al. (eds.), *Software Language Engineering - 3rd Int. Conf., SLE 2010, Eindhoven, The Netherlands, October 12-13, 2010, Revised Selected Papers*. Lecture Notes in Computer Science 6563, pp. 183–202. Springer, 2011.

[CFJ+08] P. Cuenot, P. Frey, R. Johansson, H. Lonn, M. Reiser, D. Servat, R. Koligari, D. Chen. Developing Automotive Products Using the EASTADL2, an AUTOSAR Compliant Architecture Description Language. In *Embedded Real-Time Software Conference, Toulouse, France*. 2008.

[DVR+11] J. Denil, H. Vangheluwe, P. Ramaekers, P. De Meulenaere, S. Demeyer. DEVS for AUTOSAR platform modelling. In *Proceedings of 2011 Spring Simulation Conference (SpringSim11), DEVS Symposium*. New York, NY, 2011.

[GL88] M. Gelfond, V. Lifschitz. The Stable Model Semantics for Logic Programming. In Kowalski and Bowen (eds.), *Proceedings of the Fifth Int. Conf. on Logic Programming*. Pp. 1070–1080. The MIT Press, Cambridge, Massachusetts, 1988.

[HSS+07] W. Herzner, R. Schlick, M. Schlager, B. Leiner, B. Huber, A. Balogh, G. Csertan, A. Le Guennec, T. Le Sergent, N. Suri, Others. Model-based development of distributed embedded real-time systems with the decos tool-chain. In *Procs. of 2007 SAE AeroTech Congress & Exhibition*. Citeseer, 2007.

[KNRS00] K. Keutzer, A. Newton, J. Rabaey, A. Sangiovanni-Vincentelli. System-level design: Orthogonalization of concerns and platform-based design. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 19(12):1523–1543, 2000.

[MH06] A. Metzner, C. Herde. RTSAT An Optimal and Efficient Approach to the Task Allocation Problem in Distributed Architectures. In *Real-Time Systems Symposium, 2006. RTSS'06. 27th IEEE International*. Pp. 147–158. IEEE, 2006.

[NSKB03] S. Neema, J. Sztipanovits, G. Karsai, K. Butts. Constraint-based design-space exploration and model synthesis. In *Embedded Software*. Pp. 290–305. Springer, 2003.

[Omg08] Omg. MOF Model to Text Language (MTL). Technical report, OMG, Jan. 2008.

[OPH06] R. Obermaisser, P. Peti, B. Huber. DECOS: an integrated time-triggered architecture. *E & I ELEKTROTECHNIK UND INFORMATIONSTECHNIK* 123(3):83–95, 2006.

[OSE05] OSEK. OSEK/VDX Specifications of the OS 2.2.3. 2005.

[PEP02] T. Pop, P. Eles, Z. Peng. Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems. In *Proceedings of the tenth international symposium on Hardware/software codesign*. Pp. 187–192. 2002.

[PGR+08] K. Popovici, X. Guerin, F. Rousseau, P. S. Paolucci, A. A. Jerraya. Platform-based software design flow for heterogeneous MPSoC. *ACM Transactions on Embedded Computing Systems* 7(4):1–23, July 2008.

[PM04] S. Prabhu, P. Mosterman. Model-Based Design of a Power Window System: Modeling, Simulation and Validation. In *Proceedings of IMAC-XXII: A Conference on Structural Dynamics, Society for Experimental Mechanics*. 2004.

[SBPM08] D. Steinberg, F. Budinsky, M. Paternostro, E. Merks. *EMF: Eclipse Modeling Framework (2nd Edition)*. Addison-Wesley Professional, 2 edition, Jan. 2008.

[SD07] A. Sangiovanni-Vincentelli, M. Di Natale. Embedded System Design for Automotive Applications. *Computer* 40(10):42–51, Oct. 2007.

[SG98] B. Spitznagel, D. Garlan. Architecture-based performance analysis. In *Proceedings of the 1998 Conference on Software Engineering and Knowledge Engineering*. Pp. 19–20. Citeseer, 1998.

[SHL10] B. Schatz, F. Hölzl, T. Lundkvist. Design-Space Exploration through Constraint-Based Model-Transformation. In *2010 17th IEEE International Conference and Workshops on Engineering of Computer-Based Systems*. Pp. 173–182. IEEE, 2010.

[Sin07] O. Sinnen. *Task scheduling for parallel systems*. Wiley-Interscience, 2007.

[SM01] A. Sangiovanni-Vincentelli, G. Martin. Platform-based design and software design methodology for embedded systems. *IEEE Design & Test of Computers* 18(6):23–33, 2001.

[VD02] H. Vangheluwe, J. De Lara. MULTI-PARADIGM MODELLING AND SIMULATION. In *AI, Simulation and Planning in High Autonomy Systems*. 2002.

[ZZDS07] W. Zheng, Q. Zhu, M. Di Natale, A. Sangiovanni-Vincentelli. Definition of Task Allocation and Priority Assignment in Hard Real-Time Distributed Systems. *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, pp. 161–170, Dec. 2007.