Towards Dynamic Deployment Calculation for Extensible Systems using SMT-Solvers

Klaus Becker, Sebastian Voss
Software and Systems Engineering
fortiss GmbH
Guerickestr. 25, 80805 Munich, Germany
{becker, voss}@fortiss.org

***Abstract*—During design of distributed embedded systems, the determination of the deployment of software components to execution units is a crucial subtask of the design space exploration. In static systems, the deployment can be determined at design time. However, in many cases it is desired to add new functional features into existing systems after sale. In this case, new software components have to be integrated into the former system and hence, into an existing deployment.**

We aim in a self-configuring system that ensures the integrity of the integration of new components autonomously. Our proposed process of integrating new components into a given system consists of several steps intended to be applied at run-time while the system is in operation. Beside others, the process includes a logical admission control, followed by a self-configuration of the system.

In this paper, we focus on extending an existent deployment during the self-configuration phase incrementally. We sketch a mechanism that extends existing deployments with additional components in an efficient way by using SMT-solvers. We also present an example that demonstrates how these solutions are calculated based on given deployment-constraints.

I. Introduction and Motivation

Maintainability and extensibility are important properties of long living systems, especially also in distributed embedded systems. After a system is taken into operation, there might arise several reasons to maintain the system in a functional or non-functional manner. Non-functional maintenance tasks change the system without changing the systems functional features experienced by the user. Functional maintenance affects the user-experience, e.g. by updating or extending the system functionality. Updates or extensions might be required in case of changing or new requirements, or due to changes in the environment. In this paper, we focus on extending functionality of component based systems without requiring a temporal shutdown of the system under maintenance.

In this paper, we tackle the problem of extending distributed component-based embedded systems with new components that realize new functional features. We focus on the design space exploration for the new components, more precisely on the determination of the deployment. During deployment determination it is defined which software component is executed on which execution unit, while considering different constraints that must be hold by the deployment in order to be valid. This is just one sub-step of the integration process of new components. We developed a deployment calculator which is based on the usage of a SMT-Solver.

We aim in executing these techniques at runtime by the system under maintenance itself, in order to let the system have control about its own integrity. This is for instance useful for distributedly developed systems, in which no central authority performs and proofs the integration of new functional features, but also for systems that cannot be taken out of operation completely in order to install new functionalities. Examples of such systems are production lines where each production stop denotes huge costs, remotely maintained systems which cannot be accessed by humans, but also future automobiles in which new functional features are desired to be installed after sale without having to go to a workshop.

We show a methodology to find deployments in a incremental manner. Incremental means that an existing deployment is extended with new components. The requirements are that the former system components should not be affected by the new components negatively, as well as that the delta between the configuration of the old and the new extended system should be as small as possible. In case of deployment, this means that the deployment of existing components should not change when new components are added. This is to avoid on-line migrations of components. Between the extension phases, the system operates in a static manner.

The remainder of this paper is as follows. In section II, we show the big picture on our work, which is the intended process for integrating new functions into existing systems. As our integration process is supported by properties of the underlying platform on which it is performed, we discuss these properties briefly in section III. In section IV, we show briefly how the components are designed that are intended to be added to the system. Section V shows then a sketch for an incremental deployment calculation during the self-configuration phase. This is also shown by an example with some example deployment constraints. Finally, section VI discusses related work and VII the conclusion and future work.

II. The process of integrating new functionality

We aim in an integration process including an on-line admission control for the new functionality followed by a self-configuration. During integration, it has to be ensured that the system keeps operating conform to its specification. This

is especially important for safety-critical real-time systems, because a loss of integrity might cause hazardous damage to material and life. To provide new system functionality in a plug-and-play manner, the system has to be able to verify autonomously if the new functionality can be integrated or not. However, as mentioned above, in this paper we focus on the deployment calculation which is applied during the self-configuration phase of the integration process.

Fig. 1 shows our intended five main steps that are required to integrate new software components into a given system.

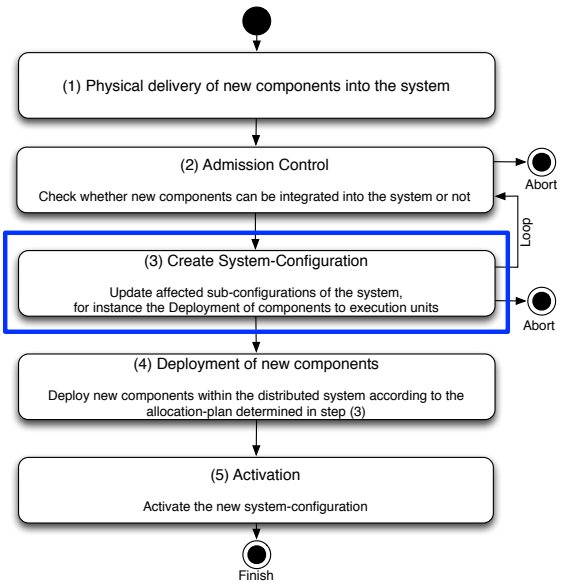


Fig. 1: Activities during the extension of the system

In phase (1), the new components are physically made available to the system. Phase (2) performs a logical admission control. Here, different formal analysis methods can be applied, which are normally applied at design time of static systems, e.g. to check interaction between components. Also compatibility and dependencies on feature level can be checked here, as well as guaranteeing security by checking certificates. However, this phase is out of scope of this paper.

If phase (2) successes, the new components can be added from logical point of view. Hence, a new system configuration has to be determined in phase (3) including the new components. During this, the new components are for instance integrated into the execution and communication schedules as well as into the deployment-configuration of the system. The latter point is in scope of this paper. We aim in an incremental self-configuration approach in which the old system configuration parts keep as most unchanged as possible (cf. section V).

However, it might happen that the self-configuration is not successful. In such situations, new components can only be integrated when a subset of the existing components is either removed, degraded or migrated to another execution node in order to free enough resources to enable a valid configuration together with the new components. This is done by going back to the logical analyses phase to select a replacement strategy for existing components. However, such replacement strategies are out of scope of this paper.

After the new deployment-configuration has been determined, the new components are physically deployed in phase (4) to the target execution units. Finally, phase (5) activates the new configuration, meaning to switch to the new schedules and hereby to activate the new components. The question of how to activate the new configuration safely without negatively affecting system service is not discussed in this paper.

All this should be possible for components that were unknown at the design time of the former system. The new components should not need to know something in advance about the system in which they are desired to be integrated. This allows highest flexibility in component composition.

III. Assumed properties of the underlying system

Our presented approach for extending systems at runtime requires support by the underlying platform. We assume some fundamental architectural drivers that are common principles.

We assume a distributed embedded system with a middleware driven distributedly accessible *data-pool* that allows indirect access to sensors and actuators. The middleware also ensures portability of the components. The portability together with the data-pool allows freedom on the allocation of the software components to the execution units, which is essential for the deployment determination.

The middleware is responsible for the transmission of sensor data into the data-pool, providing required data to the software components and delivering output data to their destinations, like physical actuators. This means, sensors and actuators are decoupled from control functions via the data-pool.

Data-dependencies are not modeled by explicit channels between components, but by specifying the required respectively provided data. The middleware has a mechanism that determines possible matches of data producers and data consumers and creates channels between the components during configuration phase, based on the data-specifications. This follows the laws of *blind communication* [7]. Due to this, components are fully exchangeable by other components that produce and require data with conforming specification. We assume also a flexible network that allows to add new network packages at runtime.

However, the concrete realization of the mentioned middleware is not further discussed in this paper as it is out of scope.

IV. Application design

The on-line integration process requires some additional pieces of information about functional and non-functional properties of the components, required for admission control decisions and self-configuration. In classical static systems, these information are only required at design time. However, in a dynamically extensible system, these information are also required at runtime during the integration process. Hence, components need to be enriched with a set of information about their functional and non-functional properties. This can for example be addressed using rich components [1].

Each component contributes to realize one or more functional features of the system. We consider components as black-boxes that might however have nested invisible sub-components. Black-box components specify their external communication by defining the required or provided data at their ports. The wiring of the communication channels between black-box components is done during the integration process according to the given data specifications at the ports.

V. On-line design space exploration for self-configuration

We consider an *incremental* self-configuration approach to integrate new components into a given configuration. This comprises that the former system configuration should keep as most unchanged as possible, meaning that existing components keep their locations and new components are deployed into free gaps. It is not desired to migrate existing components between execution units at run-time. The proposed incremental approach targets for reaching the desired level of extensibility by using on-line design space exploration mechanisms.

A. Constraints for the Design Space

System design affects temporal and spacial issues. Therefore, we distinct *partitioning*, determining the borders of a black-box component during design time, and *allocation* (aka mapping or deployment), which means deciding the assignment of software components to hardware execution nodes and pertains to spacial requirements. Based on this, a temporal configuration is the execution order of these software components (or *tasks*) on their allocated execution nodes (aka schedule) as well as the temporal order of communications (or *messages*) on a shared communication medium.

In order to solve this kind of problem, different sets of constraints need to be considered. First of all, constraints with respect to a suitable *deployment* are considered. The allocation has to comply to the existing resource constraints of the system. For instance, software components might have allocation constraints w.r.t. a dedicated location. One further constraint might be that it is desired to partition communicative component-clusters onto the same execution nodes, in order to reduce the network load. All these constraints might conclude in a *multi-objective* optimization problem with contradicting objectives.

B. Self-Configuration Process

We work towards a *hierarchically* coordinated self-configuration process, enabled by the data-pool (cf. Sec. III) that provides all required pieces of information (cf. Sec. IV) and can be used to deploy the new configuration towards the distributed system nodes. Hierarchical means that there exists a master control instance for the integration process, which cooperates with the distributed system nodes in order to determine a valid holistic system configuration.

As this approach should work on-line, we propose to have *scalable* techniques for calculating new configurations. An approach may rely on a symbolic encoding scheme for the problem under consideration. Therefore, we describe it as a satisfiability problem using boolean formulas and linear arithmetic constraints. A state-of-the-art *SAT modulo theory (SMT)* solver is used to compute new configurations for such systems in a scalable manner. Satisfiability Modulo Theory (SMT) enables checking the satisfiability of logical formulas over one or more theories. The solver proves a model as a single solution. However, optimized solutions may be of a particular interest. Finding optimized solutions takes more time and requires potentially some meta-search techniques (e.g. binary search, generic algorithms) on top of the SMT-based problem [9].

C. Deployment Problem

As described in section V-A, the calculation of a valid deployment comprises the assignment of a set of software components S to a set of execution nodes E, while fulfilling all given constraints. Furthermore, we target the reduction of required network traffic introduced by communication channels C between software components.

Our system model $\mathcal{M} = \langle S, C, E, \alpha \rangle$ contains a set of software components (SWCs) $S = \{s_1, s_2, ..., s_m\}$, a set of directed communication channels $C = S \times S$ between software components, a set of execution nodes $E = \{e_1, e_2, ..., e_k\}$, and an allocation $\alpha : S \rightarrow E$ that returns the set of execution nodes $e \in E$ to which a software component $s \in S$ is deployed. This can also be written as an allocation matrix $\alpha(s_i, e_j)$ returning 1 if $e_j \in \alpha(s_i)$, otherwise 0.

Furthermore, we define the following parameters for the system model artifacts:
$wcet : S \rightarrow \mathbb{N}$ defines the worst-case execution time (WCET) of software components $s \in S$,
$weight : C \rightarrow \mathbb{N}$ defines weights for communication channels $c \in C$, and
$tbudget : E \rightarrow \mathbb{N}$ corresponds to the time-budget of the $e \in E$.

A channel-weight $weight(c)$ is the communication load in bits/s introduced by the channel $c \in C$. The $tbudget(e)$ indicates how much time in ms is available to execute software components (in a given time period) on the given execution node. All these parameters are set by constraints to fixed constants regarding to a certain system model.

We assume the following further deployment constraints:

1) The sum of execution times $wcet(s)$ of SWCs deployed to the same execution node is not allowed to exceed the provided time budget $tbudget(e)$ of that execution node. $\forall e_j \in E \left(\sum_{s_i \in S} (\alpha(s_i, e_j) \cdot wcet(s_i)) \leq tbudget(e_j) \right)$
2) The sum of channel weights $weight(c)$ between SWCs allocated to different execution nodes E must not exceed a specified network threshold N^{Th}, defining the upper limit for the weight of network communication.

$$\sum_{c_i(s_k, s_l) \in C \mid \alpha(s_k) \neq \alpha(s_l)} weight(s_k, s_l) \leq N^{Th}$$

These constraints can be encoded into SMT formulas. The objective is to find a valid allocation α of all SWCs to the execution nodes, fulfilling all given constraints.

We encoded the parameters and constrains for the Z3 theorem prover [2]. However, the approach is not dependent on this specific SMT solver, also other solvers can be used.

D. Solution Model

The purpose of a SMT solver is to check the satisfiability of logical formulas over one or more theories. In our case, the provided solution model is a valid allocation α for the given deployment problem. Thus, the SMT solver returns one solution that fulfills the defined constraints. This is in general not an optimized solution regarding to some objective function, but just a valid solution for the specified constraints. The solution model consists of interpretations for the variables, functions and predicate symbols that makes the formula true. In our case, this gives a valid allocation matrix.

E. Example

Let the software components S and execution nodes E in the example have the following properties:
$S = \{s0, s1, s2, s3\}$
$E = \{e0, e1\}$
$wcet(S) = \{4, 4, 4, 4\}$
$tbudget(E) = \{10, 10\}$
$weight(s0, s1) = 1$
$weight(s0, s2) = 2$
$weight(s1, s2) = 4$

This is encoded as input for the SMT solver, together with a threshold N^{Th} for the maximum allowed network traffic.

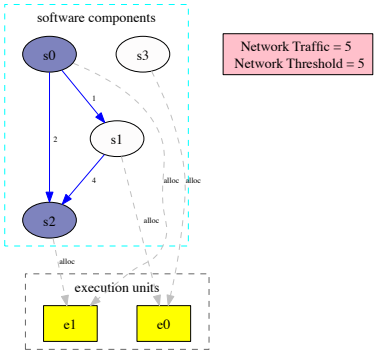
For $N^{Th} = 5$, a valid solution for deployment α is shown in Fig. 2a. Fig. 2b shows a solution for $N^{Th} = 4$, which can be hold only with a different deployment. The deployment for $N^{Th} = 3$ is the same as in Fig. 2b. A deployment for $N^{Th} = 2$ is not feasible. To minimize the network traffic, we solved the problem multiple times with decreasing N^{Th}.

F. Extending the Example

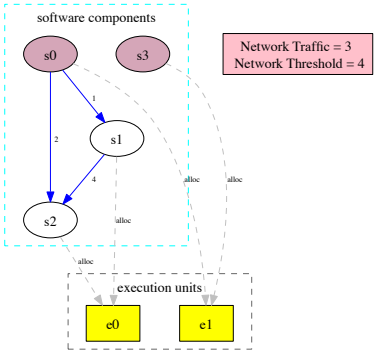
After the deployment of the initial system has been determined, the deployment should now be extended by an additional software component $s4$, having a WCET of $2ms$ and required and provided data-specifications that force the creation of two additional channels:
$weight(s0, s4) = 1$
$weight(s4, s3) = 1$

Fig. 2c shows the deployment after the new component $s4$ has been integrated. Notice that the deployment of the existing components keep untouched. This is reached by setting the former deployment of the existing components as fixed solution constraints during the deployment calculation of the extended system.

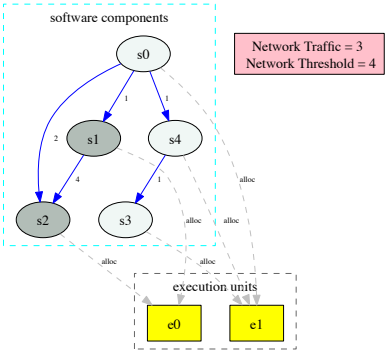
However, it might happen that the network threshold cannot be hold for the extended system. In this case, N^{Th} has to be relaxed until a valid solution is found.



(a) Solution for a deployment of 4 SWCs with 3 channels to 2 execution units, Network Traffic Threshold = 5



(b) Solution for a deployment of 4 SWCs with 3 channels to 2 execution units, Network Traffic Threshold = 4



(c) Solution in case an additional component was plugged in causing two new channels, Network Traffic Threshold = 4

Fig. 2: Solutions for example deployment problem

All the figures were generated by our deployment calculator by using the graphviz framework (www.graphviz.org).

VI. Related Work

The problem of finding optimized allocations of functions onto execution platforms (e.g. electronic control units) has for instance been considered in the following works.

In [3], an approach for centralized self-management with focus on self-configuration and self-healing in heterogeneous systems is proposed for the automotive domain. The approach uses publish/subscribe and request/response communication. As use cases, updating, installing and removing of applications

are mentioned, as well as attaching and detaching platforms. Tackled self-configuration problems are the deployment of applications (resp. their components) to heterogeneous platforms. The Self-Configuration is performed by using constraint satisfaction problems (CSP). The Web ontology language (OWL) is used to describe platforms and components with information, required by the self-configuration. Two self-configuration algorithms are presented and compared by simulation, namely *backtracking* (worst-fit) and *Iterative repair* (min-conflict). The former algorithm is slower but better usable for building configurations from scratch, while the latter algorithm is faster, independent of the number of components and better usable if a configuration for the previous system state is given.

In the project DySCAS, an automotive embedded middleware supporting extensibility was investigated. Mentioned use-cases were attaching new devices like sensors/actuators, integrating new software functionality and the shutdown of non demanded devices for power saving reasons. In case of an addition of a new task to the system, also re-allocations of existing tasks may be performed. During the deployment calculation, the aim is to maximize the total quality-of-service benefit of the tasks relative to their resource usage [8]. However, the deployment problem was solved by an algorithm and not by a more generic SMT-solver supporting a broad set of constraints in an easily usable way.

In [10], a comparison is shown about deployment-calculation by a SAT-Solver and by the Simulated Annealing Algorithm. The result was that SAT solving scales better and is more efficient for larger sets of equations. The use-case of the shown work is to find a new valid software allocation in case of a component failure. This allocation determination has to be performed as fast as possible to heal the system quickly. However, for our work we do not see the task of creating a new configuration as time-critical itself, because we apply the self-configuration only during the integration of new functionality, what we do not consider as time-critical because the system operates stable during the determination of the new configuration.

Optimisation of the allocation of functions in vehicle networks was also investigated in [5]. Self-adaptive ant colony optimisation applied to function allocation in vehicle networks was shown in [4].

Beside the deployment calculation problem, also the determination of feasible schedules is a subtask of design space exploration. An approach for the determination of static schedules of a time-triggered network-on-chip was described in [6]. The approach performs an optimization based on an evolutionary algorithm set on top of the Z3 SMT Solver.

VII. Conclusion and future work

In this paper, we have shown a methodology towards supporting extensions to existing deployments in the use case to extend distributed systems with new components. Our approach is based on the usage of SMT-solvers and is intended to be applied at system-runtime in a self-configuring manner. We discussed the assumed underlying platform properties supporting our approach and presented a sketch example.

In order to cover all parts required to obtain a complete self-configuring integration process for new components, there are still a bunch of open issues to do. Important questions are for instance how to cut the configuration problems into sub-problems that can be solved independently or hierarchically and how to setup the architecture of the self-configuration itself to reach a scalable configuration? One question is also how to use freedom on open design decisions to obtain optimal configurations, like the choice of concrete channels between components during the integration process. Also appropriate replacement strategies are of interest for the case that the self-configuration was not successful, but the new components should be integrated nevertheless.

As future work, we are going to evaluate the efficiency and scalablility of our SMT-based self-configuration approach to different sets of software components and execution units.

Furthermore, we are going to refine the deployment problem for a new platform architecture for future electric vehicles that supports mixed-critical and fail-operational features. This platform fulfills our assumed properties and is going to support extensions in a self-configuring plug-and-play manner.

VIII. Acknowledgments

This work has been investigated in the context of the Project RACE (Robust and Reliant Automotive Computing Environment for Future eCars), supported by the German Federal Ministry of Economics and Technology (BMWi).

References

[1] W. Damm, A. Votintseva, A. Metzner, B. Josko, T. Peikenkamp, and E. Böde. Boosting re-use of embedded automotive applications through rich components. *Proc. of Foundations of Interface Technologies*, 2005.

[2] L. De Moura and N. Bjørner. Z3: An efficient smt solver. *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, 2008.

[3] M. Dinkel and U. Baumgarten. Self-configuration of vehicle systems-algorithms and simulation. In *WIT'07: Proceedings of the 4th International Workshop on Intelligent Transportation*, pages 85–91, 2007.

[4] M. Förster, B. Bickel, B. Hardung, and G. Kókai. Self-adaptive ant colony optimisation applied to function allocation in vehicle networks. In *Conference on Genetic and evolutionary computation (GECCO)*, pages 1991–1998. ACM, 2007.

[5] B. Hardung. *Optimisation of the allocation of functions in vehicle networks*. PhD thesis, University of Erlangen-Nuremberg, 2006.

[6] J. Huang, J. Blech, A. Raabe, C. Buckl, and A. Knoll. Static scheduling of a time-triggered network-on-chip based on smt solving. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012*, pages 509–514. IEEE, 2012.

[7] P. Oreizy, M. Gorlick, R. Taylor, D. Heimhigner, G. Johnson, N. Medvidovic, A. Quilici, D. Rosenblum, and A. Wolf. An architecture-based approach to self-adaptive software. *Intelligent Systems and Their Applications, IEEE*, 14(3):54–62, 1999.

[8] T. Qureshi, M. Persson, D. Chen, M. Törngren, and L. Feng. Model-based development of middleware for self-configurable embedded real-time systems: Experiences from the dyscas project. 2009.

[9] S. Voss and B. Schaetz. Deployment and scheduling synthesis for mixed-critical shared-memory applications. In *Engineering of Computer-Based Systems (ECBS)*, 2013.

[10] M. Zeller, C. Prehofer, G. Weiss, D. Eilers, and R. Knorr. Towards self-adaptation in real-time, networked systems: Efficient solving of system constraints for automotive embedded systems. In *Self-Adaptive and Self-Organizing Systems (SASO)*, pages 79–88. IEEE, 2011.