Model-Driven Development of Self-describing Components for Self-Adaptive Distributed Embedded Systems

Gereon Weiss, Klaus Becker, Benjamin Kamphausen
Fraunhofer Institute for Communication Systems ESK
Hansastraße 32, 80686 Munich, Germany
Email: {gereon.weiss, klaus.becker, benjamin.kamphausen}@esk.fraunhofer.de

Ansgar Radermacher, Sébastien Gérard
CEA LIST,
Laboratory of Model Driven Engineering for Embedded Systems
Point Courrier 94, Gif-sur-Yvette, F-91191 France
Email:{ansgar.radermacher, sebastien.gerard}@cea.fr

***Abstract*—Today's distributed embedded systems comprise various fields of application. Increasingly they are deployed in complex scenarios and must be able to adapt to changing environments and internal system changes. Such self-adaptive embedded systems pose great advantages in terms of flexibility, resource utilization, energy efficiency and robustness. The realization of these systems require enhanced development methods to incorporate the adaptation in the design. We introduce a novel concept for the model-driven development of self-adaptive embedded systems. The focus of our work is the definition and transfer of the information needed for the adaptation at runtime. This is preserved as so-called self-description of the components. We present our self-x profile, a modeling extension for describing the adaptation, and the respective design flow with built-in transformations. Furthermore, we outline the applicability of our methodology in an automotive use case.**

I. Introduction

Today's embedded software systems are applied in diverse scenarios and have to execute manifold tasks. More and more, these systems have to work dependently in changing environments and react on inner system changes. Such self-adaptive embedded systems [1] have to take adaptation decisions at runtime, with respect to changing requirements. For example, modern cars comprise various driver assistance systems, which are increasingly interacting. Traditional development for such embedded software systems focuses on implementing static systems. All system information is specified in the design and requirements are met, when the system is deployed. A main challenge for self-adaptive systems is to bring the *needed information for the adaptation* into the resource-constrained runtime. Other challenges which arise are e.g., the definition of constrained adaptation for safety-critical systems, the decision taking at runtime under changing requirements and the explosion of possible numbers of runtime configurations.

Model-driven approaches have already been applied successfully to cope with the complexity of static embedded systems. Domain-specific adoptions allow their development in distinct domains. More and more non-functional requirements are considered explicitly in the design of these systems. Since these properties are subject to change at runtime within self-adaptive systems, novel design strategies are needed to conserve and consider them at runtime. Thus, for exploiting the opportunities of model-driven development for self-adaptive embedded systems, new design approaches need to be researched, which integrate as well as possible into present methodologies.

In the area of adaptive software diverse research has been pursued [1] [2]. However, most approaches focus on general purpose computing and do not consider adaptive embedded distributed systems appropriately, e.g., with respect to resource consumption, distributed control and runtime constraints. In particular the integration into domain-specific design methodologies and automatic generation of problem-tailored components is not addressed yet. For implementing self-adaptive embedded systems, the adaptation has to be defined in the design phase and enforced at runtime. Therefore, the software has to be aware of its own structure and state. However, only a subset of the design parameters is required at runtime for supporting the adaptation decisions.

In this work, we introduce our novel design methodology for self-adaptive distributed real-time embedded systems (DRTES), which integrates with present development approaches. The required information is specified in the design phase, transferred to the runtime and made accessible via defined interfaces. We present our so-called *self-x profile* for defining the adaptation of the system and *self-descriptions* of the components. In an initial automotive case study with a partial in-vehicle networking operation, we show the applicability of our approach.

This paper is organized as follows: In Section II we outline previous work related to our approach. Afterwards, we introduce our novel model-driven approach for the development of self-adaptive embedded systems in Section III. We describe our new self-x profile capturing the adaptation information in the design. Subsequently, the provided transformations in the development are shown, enabling an iterative refinement and validation. A first use case of a partial in-vehicle network operation is outlined in Section IV, demonstrating the applicability of our approach. Finally, the paper is concluded in Section V.

II. Related Work

In the recent years, various research has been carried out for building self-adaptive systems. Since these systems have to analyze their own state, often reflection techniques are used. Reflection has different forms. As in [3], we distinguish:

- *Introspection*: The system structure can be accessed but not modified.
- *Structural Reflection*: The system structure can be dynamically modified.
- *Computational (Behavioral) Reflection*: The system semantics (behavior) can be modified.

In component-based systems *introspection* is supported for instance by the CORBA component model (CCM) [4]: A client which holds a reference to a component can obtain information about the ports that are provided by the component or retrieve a certain port via its name. This information is not provided by code written by a developer, but by a *container* that is generated from the syntactical component description. It is typically only used for requests to ports typed with a known interface. Fractal [5] provides a similar reflection mechanism: a membrane (corresponding to a container in CCM) provides references to ports owned by the component. There exists also a Lightweight CCM specification, defining a subset of the functionality of the full CORBA CCM specification.

A natural ability of component-based systems is the change of connections. For instance, CCM provides reflective methods to obtain the current connections and change their value. However, there is no support for global consistency, implying that the application logic itself must assure, e.g., that connections are not changed while requests are in progress. *Computational reflection* can be supported by means of interception, i.e. the execution of additional code before and/or after the execution of a service. In most component models, it is however limited since interception code should not reroute requests to different business logic or components. A typical application is to trace or block, e.g. assure access control or serialize concurrent computations.

Research for computational reflective automotive applications has been done by [6] to built adaptive defense software. Based on the AUTOSAR [7] middleware, the system can observe system behavior, check on-line properties and perform recovery actions. This leads to defense software that is decoupled from the real functional system.

As shown, introspection is integrated in diverse component-based approaches. However, the amount of self-information cannot be controlled. For instance, the CCM standard has a quite complete set of introspection interfaces that imply a considerable overhead. Therefore, these are removed in the lightweight variant of CCM. Depending on the kind of foreseen adaptation purpose and the available resources, a different subset of self-information is required at runtime. The following section introduces selected related work within the field of self-adaptive systems.

A. Self-Adaptive Systems

In the area of self-adaptive systems, many approaches have been pursued aiming at making systems more adaptable at runtime. In the following we outline some representative approaches.

The DIVA project [8] focuses on dynamic variability in complex systems following a component-based approach, close to Fractal. The actual configurations of the application are built at runtime, based on aspect-oriented modeling (AOM) and models at runtime. A main objective of the DIVA approach is to model adaptive systems without having to enumerate all possible configurations statically. The application is modeled using a base model, which contains the common functionalities and a set of variant models, which can be composed with the base model. An adaptation model specifies which variants should be selected at runtime according to the adaptation rules and the current context of the executing system. However, the DIVA approach is not targeting distributed embedded systems, e.g. due to the large resource consumption overhead at runtime.

Another research area for providing self-adaptive functionalities are middleware-based approaches. In the RUNES project [9] [10], a component based middleware for reconfigurable networked embedded systems and wireless sensor networks was investigated. Heterogeneity is supported by installing the middleware on top of different supported operating systems for different types of hardware platforms. The middleware overhead is claimed to be lightweight and hence applicable to resource-constrained devices. The iLAND project [11] also researches a middleware for deterministic reconfiguration of networked embedded systems. Main objectives are the support of real-time systems, service-oriented architectures and timely composition and reconfiguration of services. However, such lightweight middleware approaches do not focus on tailoring the runtime information for embedded targets. Within the MUSIC project [12] model-driven development of self-adapting applications for mobile users in ubiquitous computing environments has been researched. The DySCAS project [13] [14] focused on developing a middleware enabling dynamic self-configuration of automotive software systems. Adaptation of the system is realized through decision points, at which a policy has to be evaluated. Policy evaluation at runtime still poses a not insignificant overhead for an automotive embedded systems.

Also targeting the automotive domain, in [15], an extension to the AUTOSAR [7] framework is shown to support on-line reconfiguration on architectural level. However, only local reconfiguration is supported yet. A proposal to provide self-healing software in the automotive domain is introduced in [16], exploiting an *Organic Design Pattern* (ODP). By specifying constraints for the adapted ODP, self-healing should be performed in case of failures of automotive control units.

In contrast to the previously described work, we focus on the integrated development of self-adaptive DRTES, in particular defining and tailoring the necessary information as

self-descriptions. We also present an iterative development approach, which includes a simulation engine providing feedback for early validation and refinement.

III. Model-Driven Development of Self-Adaptive Embedded Systems

Enabling self-adaptation in distributed real-time embedded systems (DRTES) promises several enhancements, e.g., with respect to their flexibility, resource utilization, (energy-) optimization and robustness. Though, for implementing such systems, the adaptation has to be integrated into the model-driven design of nowadays embedded systems and address the characteristic challenges [17] for these systems.

The scarceness of resources in DRTES poses a major challenge to make such systems self-adaptive, because this limits the capacity to enhance the system with self-adaptation. Since these systems are developed domain-specifically for their areas of application, approaches targeting these domains should integrate with prevailing domain-specific design and targeted runtime systems. The mentioned heterogeneity of the resources often results in very diverse processing power of the system. Therefore, runtime artifacts have to be tailored for the overall system and possibly, for all of its components. Because often also safety-critical tasks or applications of mixed-criticality are executed, the degree of allowed adaptations must be confined a priori in the design. These characteristics necessitate the research of enhanced methodologies and tools for self-adaptive embedded systems, which take into account the domain-specific requirements.

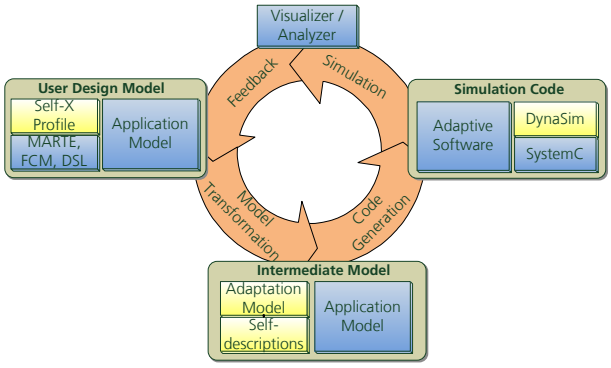

Fig. 1. Abstract view on our investigated methodology

If distinct adaptability of the software is specified during the design phase, software can react adequately at runtime e.g., to environmental changes or changing requirements. In our model-driven methodology the required information about the adaptability is added to the models during the design. Addressing DRTES we only consider *controlled adaptation*. This is realized by allowing unanticipated behavior of the system only for uncritical parts or subsystems. For critical system parts no adaptation at all or anticipated behavior is allowed. The possible adaptations are confined in so-called configuration spaces. We utilize architecture descriptions to model the system and exploit architectural adaptation [18] at runtime. Also, we investigate a new iterative model-driven design methodology, in which the software components behavior can be validated and refined already at early design phases. This refinement is based on simulation of the software execution, providing early feedback about the adaptive behavior, response times, communication overheads, etc. The required system properties are preserved by using model transformations and code generation. This procedure is shown abstractly in Fig. 1.

The design models are enriched with additional information specifying the possible space of adaptability of the developed software system. This allows to enrich (domain-specific[1]) models with adaptation as additional view. By this, the design follows the separation of concerns [19] approach with respect to adaptation as a particular concern. The adaptation information can include, e.g., valid execution platforms of a software component or possible alternatives of the software's behavior. In order to provide the required additional information for adaptability at runtime, we investigate ways to decouple the software's functional properties from this additional self-information. The latter mainly comprises so-called *non-functional properties* (NFPs). Examples for such NFPs are required resource usage (memory) or timing parameters, like activation periods. This information is needed at runtime to make adequate adaptation decisions. The possible adaptability can be expressed via variants of software components or optional features that can be de- and activated at runtime. These descriptions are included in the self-x profile, which is described in Section III-A. The development of self-describing software components for DRTES is integrated in an iterative design process as shown in Figure 1.

In order to support the development of software for different application domains, the designer has the freedom to model the software system via different appropriate domain-specific modeling extensions, in the form of UML profiles. Within the automotive domain, this may be done by applying the *Electronics Architecture and Software Technology - Architecture Description Language* (EAST-ADL) [20]. An EAST-ADL model is organized in four abstraction levels, from a very abstract upper vehicle level to an implementation specific lowest level, which builds the connection to the AUTOSAR [7] standard. The EAST-ADL is also provided as a UML [21] profile for the Eclipse Papyrus MDT project [22]. In our work, we treat the EAST-ADL as a prime example for domain-specific modeling that can be used within our design methodology. We concentrate mainly on the functional design level, which is the second lowest level and allows to specify functional software components, hardware components and their allocation relationships. The EAST-ADL complies to the AUTOSAR standard and extends this, e.g. by feature modeling including concepts to support software product lines [23]. However, these features are intended to introduce static variability at design time, but are not intended to allow dynamic adaptability at runtime.

[1]Although UML is not a domain-specific modeling language, its profiling mechanism can be used for domain-specific customizations

A. Self-X Profile

In order to introduce dynamic self-adaptive concepts into embedded software systems, we investigate a so-called *self-x profile*. As UML profile it allows to enhance existing domain-specific design models with descriptions for adaptation. It consists of:

- Self-descriptive properties of the system, which are used for determining the system state and are fundamental for planning runtime adaptation.
- Syntactical elements for the definition of the adaptivity inside the system architecture.

The self-x profile is kept domain-independent. This should ensure the applicability in different application domains of DRTES. Mainly for specifying NFPs, the UML profile MARTE [24] is used. MARTE is a standardized profile to model and analyze software for embedded real-time systems. Specifically, in our self-x profile, we use the standardized basic NFP definitions and the *value specification language* (VSL) of the MARTE standard to specify non-functional properties. The basic NFP definitions include not only values, but also units. For instance, a worst case execution time (WCET) of a task is a "NFP_duration", which has a value and a unit, e.g., could exemplary be defined as $WCET = (3, ms)$. MARTE standardizes other NFP types such as bandwidth or power consumption as well, which make the associated values exchangeable. The dependencies of the design model to MARTE, our self-x profile and other domain specific profiles are depicted in Fig. 2.

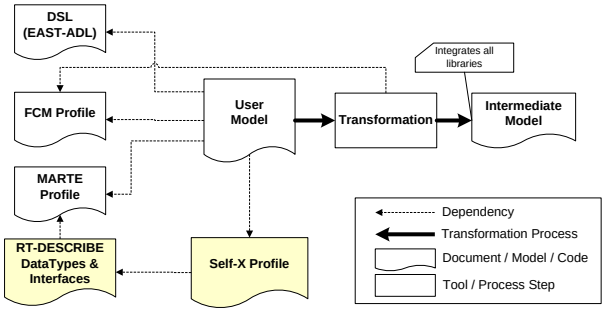

Fig. 2. Transformation steps and dependencies of our methodology

By applying the self-x profile to the user model, it can be enriched by adaptability and the additional NFP information about the software, which will be required at runtime to provide self-adaptation. The additional NFP information are held modular for each software component, building the self-descriptions of the components. The self-descriptions consist partially of static non-functional properties (NFPs), which should be specified by the system designer. In addition to these static properties, the self-description contains also dynamic runtime information. Storing the self-descriptions modularly for each component is valuable in distributed systems, because a centralized storage of the required information is inadequate. Failures inside a centralized storage could lead to a total inaccessibility of the stored information for the whole system, whereas in case of a distributed storage, only the information stored on a failing node are lost.

The information provided by the self-description can be divided into the following three categories:

1) Information that is provided by the developer via the self-x profile, e.g., information about the vendor, scheduling properties or required memory spaces. This information is definable on type and instance basis. For example, information that is identical for all instances of a component should be defined on type basis, while information that differ for the instances of a component should be defined at instance basis. During runtime, the information that is identical for all instances is only stored once per execution unit in order to save memory space.
   Each corresponding stereotype has an attribute, which is typed by a MARTE tuple-type, a specialization of a UML data-type. This allows to enter the provided information via MARTE's value specification language (VSL). For instance, an attribute can be: swcInfo={requiredRAM = {unit = KB, value = 3.3}, vendor={value="CEA+ESK"}}.
2) Information that can be derived from the model. A part of the information that may be useful at runtime is already available within the model or can be derived from it. A simple example is the information about the component ports (name, type, interfaces, etc.), its attached connectors and corresponding connected components. Thus, it is not part of the self-x stereotypes, but needs to be reified in form of a suitable instance configuration within the self-x container extension of a component.
3) Information that can only be calculated at runtime. An example for such an information is the current state of a component instance or its resource usage. Thus, this information is not stored but calculated by operations within the container extension.

The self-descriptions are available for software-components as well as hardware-components, like electronic control units and communication buses. The data stored in the stereotype attributes (or a subset of this on resource limited platforms) is provided at runtime by a container extension, introduced in section III-B.

The adaptability is expressed by *configuration spaces*, comprising sets of features and constraints for their activation. If a feature is activated, this causes the activation of its related software components. Whereas MARTE allows the definition of fixed modes with fixed transitions, configuration spaces provide a more flexible way of adaptability [25]. Configuration spaces define a part of the configuration in which a certain number of configuration choices is fixed. Other choices are left open to be decided at runtime, enabling a range of possible adaptivity. For this, the system consists of a combination of statically designed configurations and free addable features. A concrete example for such a configuration space would be

the definition of configurations for driver assistance systems in a car. Some features (like for safety reasons the electronic stability control) are always executed in the system, whereas others (like the parking assistant) are only activated when needed. In this example, the decision of activating the parking assistant is taken at runtime. Static configurations or modes, as they are also used in standards like MARTE, exhibit a disadvantage in respect to the requirements of adaptive systems. This enables only to model a distinct, predefined level of adaptivity. This denotes an incisive lack of flexibility, concerning the enormous amount of different possible contexts the targeted systems may react to.

Our approach focuses on describing the adaptability of the system on a more abstract level. Therefore, we define a lean feature model. The adaptive features defined are implemented by certain functionalities of the system. For instance a feature ”parking assistant” is an abstract mapping of all components, needed to build the respective functionality. Whereas this leads in a traditional system to a fully definition of the systems functional characteristics, we differentiate between optional and mandatory features in the system in order to distinguish adaptive parts from a static base. Furthermore, dependencies and constraints are defined which are necessary for the system to stay in a consistent state. Thus, in a self-adaptive system as considered here, it is possible that two features are incompatible because of their internal realization. Therefore, priorities for features to resolve such conflicts through the runtime environment are introduced. As mentioned before, the flexibility of our approach increases with the combination of configurations and the previously described features. Therefore we add a second more abstract layer into the model, which is leveled above our feature model. The so-called configuration spaces define several configurations which are assigned to specific contexts. Every configuration consists of a group of features and optionally designed sub-configurations. By this contexts are indirectly connected to a defined combination of configurations, which leads to the activation of the specified features. At this point, the information about constraints on the feature connection is important. For instance, if two configurations should be activated, containing two mutually exclusive features, the adaptation mechanism has to resolve the emerging conflict at runtime. This information can be derived completely from the model, whereas this conflict can be resolved either during deployment or left unresolved to be carried out at runtime.

B. Transformations and Iterative Design with Simulations

Because computational power and memory space are limited inside the desired application domain of embedded systems, a usage of the complete design model at runtime is inadequate. Instead, we investigate a methodology to provide a problem-suited transformation from the design model to relatively small intermediate models, containing only the required data for the current purpose. This reduces the required amount of data to be stored at runtime.

As we follow the approach of separating the adaptation in the design, we investigate a mechanism to decouple the functional properties from the NFPs. A composition of them allows a more flexible way of adaption. In order to support the decoupling, the design model is enriched during the transformation step with containers encapsulating the self-description, thus providing a subset of the model information at runtime.

For generating these intermediate models, we investigate a model-to-model transformation procedure, which converts the design model into various problem-suited intermediate models, each supplemented by the self-descriptions and information about adaptability. Depending on the purpose and intended degree of self-adaptation, like for self-configuration or partial (de-)/activation of functions, the intermediate models may differ, e.g., in the amount, expressiveness, and encoding of their self-describing attributes. In our investigated tool chain, we use the *embedded Component Container Connector Model* (eC3M) [26] [27] to transform the design model into the intermediate models. The following Fig. 3 shows such a transformation and the resulting container. Because eC3M is

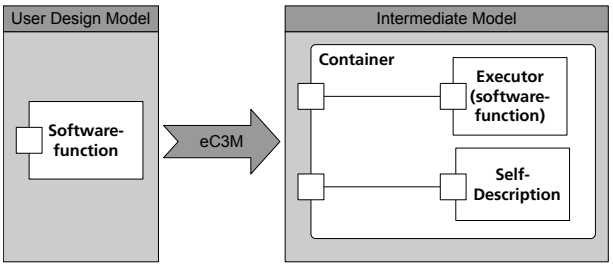

Fig. 3. Container with self-description, generated by eC3M

based on the *Flex-eWare Component Model* (FCM) [28], the design model stereotypes are mapped to appropriate FCM stereotypes. FCM is a component model developed to unify different component model concepts such as Fractal [5] or the CORBA component model (CCM) [29]. The advantage of using eC3M compared to other related tools is that it is well integrated into the UML modeler Papyrus [30] and the flexibility of FCM allows for integration with components models adapted for a certain domain. This implies that the user can work on his domain model while eC3M synchronizes domain model information with FCM, e.g. uses the appropriate FCM representation of a MARTE *RtUnit* or a MARTE *ClientServer port* as described in [31].

Fig. 4 shows the mapping of basic NFP data types, which are part of the self-description in the target implementation. The MARTE data-types are strings (that respect the VSL grammar), which can neither be stored nor evaluated efficiently. For instance, a *NFP_DataSize* type can be expressed using different units. On a target, the unit should be fixed (in this case to bytes) which implies that it is sufficient to store the numerical value in a suitable format (e.g., a long in case of the data size).

Technically, the mapping is captured in UML by means of a composite class, in which an element of the development

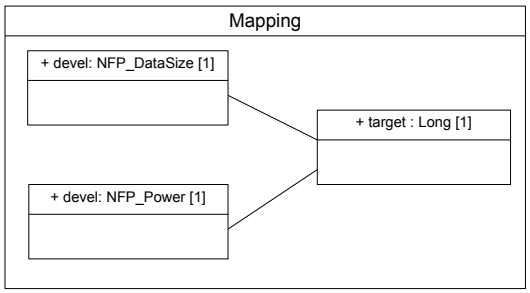

Fig. 4. Mapping of basic NFP types to a target

model is connected with an element of the target model. The mapping is recursive: a relatively complex type such as the MARTE arrival pattern might be mapped to a simpler variant only supporting periodic patterns. For instance, the mapping of the contained attributes of type *NFP_Duration* is not specified in the arrival pattern mapping, but automatically captured by mappings of the contained types that have already been defined.

In order to arm the designer with an early feedback about the system behavior, our methodology also includes simulations of the application at early design stages. The simulations offer validations of the adaptive behavior and provide feedback to the designer for an early refinement of the system. This refinement may happen in an iterative manner via multiple executions of the simulation. We investigate a SystemC [32] based simulation framework called DynaSim [33], currently compatible to automotive applications modeled in EAST-ADL. The feedback is stored in an analysis model and may be integrated into the design model manually or in a semi-automatic way. The following Fig. 5 shows this feedback loop, exploiting different model transformations. It includes the transformation to DynaSim-specific meta-models for the input and analysis. The feedback loop is intended to be executed multiple times during design phase, producing an iterative refinement of the design model.

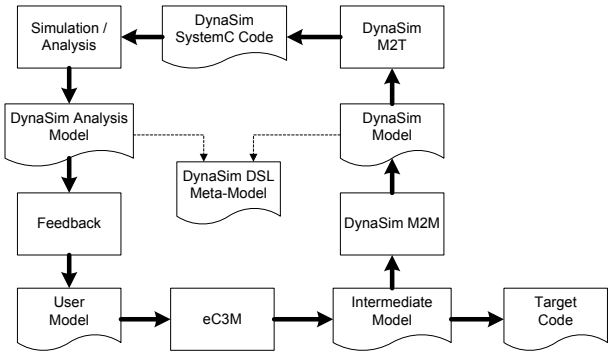

Fig. 5. Transformation steps of our methodology plus feedback loop

This section has given an overview on the modeling approach and design process intended in our investigations. However, the work has not been finished yet and the concepts, e.g., implementing the feedback loop, will be refined in future work. The next section introduces the application of self-adaptation in the automotive domain.

IV. Automotive Use Case

With our approach we mainly focus on DRTES in the transportation domain, but also other areas may benefit from our approach. For the automotive domain we investigated several runtime use cases to which our new methodology is applicable.

Today's automobile electronic systems are mostly designed statically and pre-configured before the delivery of a car. Typically, the initial configuration is implemented at the end of the production line by programming the flash memory of the Electronic Control Units (ECUs). Whereas this allows setting up previous validated configurations, the complexity of defining configurations is increasing with every newly added feature. For facing this complexity problem, automotive embedded systems may be enhanced by self-adaptation capabilities. During runtime, the self-adaptive automotive system can start and stop functions, as needed in the present driving situation. For instance, a parking assistant is executed only while the car is actually parking and the cruise control is only loaded on a highway drive. This allows for a maximized resource utilization and enables energy-efficiency or even a reduction of costs through saving hardware resources is possible. If the system is clustered in such a way that unnecessary functionality in distinct situations is allocated to a single part of the network, this part of the in-vehicle network can be shut down resulting in saving energy. This mechanism is called *partial in-vehicle networking*. As these applications give a notion of the manifold advantages of self-adaptation within embedded systems, here exemplary represented by the automotive domain, many challenges have to be addressed - especially for the application in DRTES.

A. Automotive Use Case Exterior Light

For evaluation purposes of our approach we adopt a characteristic automotive application within the car-body domain. This application comprises four interacting applications of an automobile: *exterior light*, *direction indication*, *central door locking*, and *keyless door entry*. The sensors, controls and actuators necessary for a typical realization of these applications are depicted in Figure 6. The exterior light feature allows controlling the front and rear lights of the vehicle. The lights can be switched on/off manually via the light switch or automatically through darkness or rain, detected by the rain/light sensor. These inputs are interpreted by the function exterior light control which switches the front light unit (left/right) and rear light unit (left/right). For the direction indication a direction indication switch can be used to signal the turning direction. With the hazard light switch, risky driving situations can be signaled to other road users. Therefore, the direction indication master control informs the direction indication front and rear controls about the designated status of the direction indication lights. These turn the direction indication lights on or off in the front and rear light units. Central door locking

allows locking and unlocking all doors simultaneously by using the key in the lock or by radio transmission. A radio receiver signals the information to the central door locking control. This function flashes the direction indication lights for a feedback to the driver and controls the four door locks of the car. An additional feature to the un-/locking of an automobile is the keyless entry. A driver can approach his car with the key in his pocket and the doors will unlock automatically. It can be locked by simply pressing a button on the door handle. Antenna components detect the key in the surrounding and inform the central door locking function which in turn unlocks the doors.

The deployment of the software components to the ECUs can be defined quite flexible in our approach. Several allowed ECUs can be defined for each SWC, allowing migrations of a SWC between the set of its defined ECUs at runtime.

B. Scenario of Partial In-Vehicle Network Operation

A scenario which can be enabled by self-describing components is the partial in-vehicle network operation. In this use case it is possible to shut down certain parts of the in-vehicle network or single ECUs, to save resources (e.g. energy) during runtime. This can be done in certain contexts, e.g., when all functions located in a distinct area of the network are not required, or can be substituted by functions running on other platforms. These functions might be started dynamically or, for simplicity reasons, run as shadow tasks in the background all the time. The potential benefits of a partial in-vehicle network operation, of course, strongly depends on the mapping of software functions to the ECUs. For an optimal allocation, in terms of the partial in-vehicle network operation, the distribution should cluster functionality which is and which is not used in the same context.

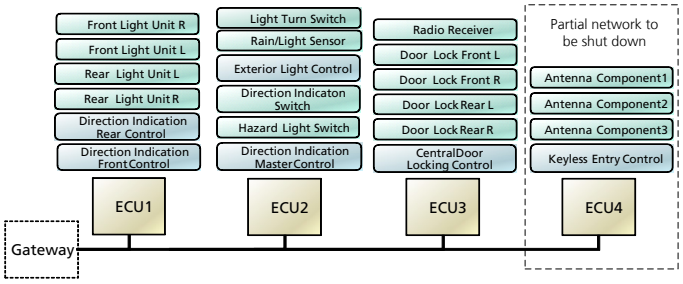

Fig. 6. Overview of the partial in-vehicle network operation scenario

Figure 6 depicts a vehicle system consisting of four ECUs, connected by a *Controller Area Network (CAN)*. Sensors, actuators and software functions build the functionality "Exterior Light" (as described in Section IV-A). Also, the allocations of the software functions, sensors and actuators to the respective ECUs are shown. With regard to the partial network operation, the allocation of the system is essential for the potential to shut down certain parts of the in-vehicle network. Thus, the software functions and the sensors/actuators are deployed in a suitable way to permit shutting down certain parts of the network in distinct contexts. In the shown scenario for instance, the functionalities for the keyless entry feature are executed on ECU4. This feature allows the driver to lock/unlock the car when leaving/approaching the car, while keeping the key in the pocket. When a car is parked, this allows a convenient and safe access to the car. However, once the driver got into the vehicle this feature is actually not needed anymore. With respect to the allocation, this implies in our example that ECU4 can be shut down completely (cf. Figure 6) in other contexts (e.g. while driving). Only the network traffic on the CAN bus has to be kept alive. When the situation changes and the keyless entry feature has to be provided again, ECU4 and the respective software functions allocated to it can be activated again. This dynamic activation and deactivation of in-vehicle systems allow saving runtime resources, like energy and processing power. We have applied our concepts for

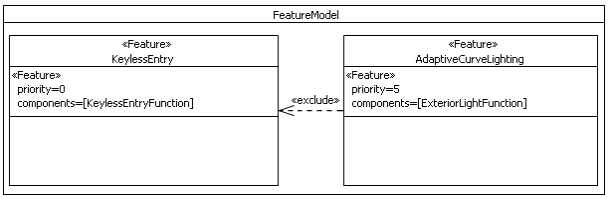

Fig. 7. Mutually excluding features in the partial networking scenario

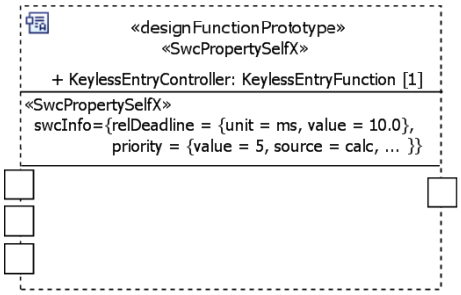

Fig. 8. Component modeled with self-description

modeling this scenario (cf. Section III). The system is designed exploiting EAST-ADL and its respective UML profile. The exterior light use case was modeled at the functional design architecture level of EAST-ADL, where allocations to ECUs are defined. For specifying the necessary information for the adaptation we have applied our self-x profile. As is exemplary shown in Figure 7, the adaptive feature keyless entry is modeled here to be explicitly excluded by another feature for adaptive curve light. This defines the need for deactivating the keyless entry feature adaptively, when the adaptive curve light is activated (denoting in this example another context, like driving). In contrast to static features, e.g. in software product lines, these adaptive features are selected at runtime. The stereotype for specifying features is applied from the self-x profile. In this example it includes attributes for referencing the feature realizing components and priorities for runtime feature conflict management. Self-descriptions for the components have been defined utilizing the self-x profile extension.

A sample for such a self-description is depicted in Figure 8 for the *Keyless Entry Controller* component. The shown class is a functional component in the EAST-ADL functional design architecture, specified as *designFunctionPrototype*. It is enhanced by the *SwcPropertySelfX* stereotype which defines a *swcInfo* attribute. The latter describes the characteristics of the functions in a MARTE VSL string. With the self-x profile, it is possible to specify the information for the self-adaptation of the components in the partial network operation scenario. Presently, our approach is applied for this limited automobile scenario but future work will be exploring other automotive scenarios and defining concrete mappings of self-descriptions.

V. Conclusions

Distributed self-adaptive embedded systems are promising for manifold application areas, but new challenges for their development have to be solved. We have presented our novel concepts for the model-driven development of such systems. Our approach focuses on the definition and transfer of the necessary information for the adaptation. It is automatically embedded as self-description in the components. Furthermore, we have described the transformations involved in our iterate design flow. Through an iterative development with simulations and feedback to the models, the adaptation can be defined and a refinement of the design is possible. Also, we have applied our concepts in a use case of a self-adaptive automobile software system. Future work within this project will be the refinement of the methodology and the integration of the automatic simulation feedback for the iterative model refinement.

Acknowledgments

This work was supported by the Programme Inter Carnot Fraunhofer from the German Federal Ministry of Education and Research (BMBF) (Grant 01SF0804) and the French L'Agence nationale de la recherche (ANR) within the project RT-DESCRIBE.

References

[1] B. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic *et al.*, "Software engineering for self-adaptive systems: A research roadmap," *Software Engineering for Self-Adaptive Systems*, pp. 1–26, 2009.

[2] M. Salehie and L. Tahvildari, "Self-Adaptive software: Landscape and Research Challenges," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 4, no. 2, pp. 1–42, 2009.

[3] F. Ortin and J. Cueva, "Non-restrictive computational reflection," *Computer Standards & Interfaces*, vol. 25, no. 3, pp. 241–251, 2003.

[4] OMG, *CORBA Component Model Specification, Version 4 (part of the CORBA 3.1 Specification)*, 2008, oMG Document formal/2008-01-08.

[5] The Fractal Project, http://fractal.ow2.org.

[6] J. Fabre, M. Killijian, and F. Taiani, "Robustness of Automotive Applications Using Reflective Computing: Lessons learnt," 2011.

[7] "Automotive open system architecture," http://www.autosar.org.

[8] B. Morin, O. Barais, and J. Jézéquel, "K@RT: An Aspect-Oriented and Model-Oriented Framework for Dynamic Software Product Lines," in *3rd International Workshop on Models@ Runtime, at MoDELS*, 2008.

[9] P. Costa, G. Coulson, C. Mascolo, G. Picco, and S. Zachariadis, "The RUNES middleware: A reconfigurable component-based approach to networked embedded systems," in *Personal, Indoor and Mobile Radio Communications, 2005. PIMRC 2005. IEEE 16th International Symposium on*, vol. 2. IEEE, 2005, pp. 806–810.

[10] F. Oldewurtel, J. Riihijärvi, K. Rerkrai, and P. Mähönen, "The RUNES Architecture for Reconfigurable Embedded and Sensor Networks," in *2009 Third International Conference on Sensor Technologies and Applications*. IEEE, 2009, pp. 109–116.

[11] M. García-Valls, I. Rodríguez-López, L. Fernández-Villar, I. Estévez-Ayres, and P. Basanta-Val, "Towards a middleware architecture for deterministic reconfiguration of service-based networked applications," in *Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on*. IEEE, 2010, pp. 1–4.

[12] M. Khan, R. Reichle, and K. Geihs, "Architectural constraints in the model-driven development of self-adaptive applications," *Distributed Systems Online, IEEE*, vol. 9, no. 7, p. 1, july 2008.

[13] I. Jahnich, I. Podolski, and A. Rettberg, "Towards a middleware approach for a self-configurable automotive embedded system," *Software Technologies for Embedded and Ubiquitous Systems*, pp. 55–65, 2008.

[14] A. Rettberg, R. Anthony, D. Chen, I. Jahnich, G. de Boer, and C. Ekelin, "A dynamically reconfigurable automotive control system architecture," in *Proceedings of the 17th IFAC World Congress, 2008*, vol. 17, 2008.

[15] B. Becker, H. Giese, S. Neumann, M. Schenck, and A. Treffer, "Model-based extension of autosar for architectural online reconfiguration," *Models in Software Engineering*, pp. 83–97, 2010.

[16] H. Seebach, F. Nafz, J. Holtmann, J. Meyer, M. Tichy, W. Reif, and W. Schäfer, "Designing self-healing in automotive systems," *Autonomic and Trusted Computing*, pp. 47–61, 2010.

[17] G. Weiss, K. Becker, A. Radermacher, and S. Gérard, "RT-DESCRIBE: Self-describing Components for Self-Adaptive Distributed Embedded Systems," *3rd Workshop on Adaptive and Reconfigurable Embedded Systems (APRES 2011)*, 2011.

[18] D. Sykes, W. Heaven, J. Magee, and J. Kramer, "Exploiting non-functional preferences in architectural adaptation for self-managed systems," in *Proceedings of the 2010 ACM Symposium on Applied Computing (SAC)*, 2010, pp. 431–438.

[19] W. Hürsch and C. Lopes, "Separation of Concerns," *College of Computer Science, Northeastern University, Boston*, 1995.

[20] ATESST2, "EAST-ADL Domain Model Specification 2.1 RC3," http://www.atesst.org, June 2010.

[21] OMG, "Unified modeling language (uml)," http://www.uml.org.

[22] Papyrus MDT project, http://www.eclipse.org/papyrus.

[23] P. Cuenot, P. Frey, R. Johansson, H. Lönn, M. Reiser, D. Servat, R. Koligari, and D. Chen, "Developing automotive products using the EAST-ADL2, and Autosar compliant architecture description language," in *Embedded Real-Time Software Conference, Toulouse, France*, 2008.

[24] OMG, "Marte specification version 1.0," http://www.omg.org/spec/MARTE/1.0/PDF, November 2009.

[25] M. Zeller, G. Weiss, D. Eilers, and R. Knorr, "An approach for providing dependable self-adaptation in distributed embedded systems," in *Proceedings of the 26th ACM Symposium on Applied Computing (SAC)*, 2011.

[26] CEA, "eC3M: embedded Component Container Connector Model / Middleware," http://www.ec3m.net.

[27] M. Fredj, A. Radermacher, S. Gerard, and F. Terrier, "eC3M: Optimized model-based code generation for embedded distributed software systems," in *10th Annual International Conference on New Technologies of Distributed Systems (NOTERE)*. IEEE, 2010, pp. 279–284.

[28] Flex-eWare project, http://www.flex-eware.org.

[29] OMG, "CORBA Component Model (CCM)," http://www.omg.org/technology/documents/.

[30] Fredj, M. and Radermacher, A. and Gerard, S. and Terrier, F., "A Developer Oriented View of Component-Based Embedded Systems," in *Software Engineering and Advanced Applications (SEAA), EUROMICRO Conference*. IEEE, 2010, pp. 417–424.

[31] A. Radermacher, A. Cuccuru, S. Gerard, and F. Terrier, "Generating Execution Infrastructures for Component-oriented Specifications With a Model Driven Toolchain – A case study for MARTE's GCM and real-time annotation," in *Eighth International Conference on Generative Programming and Component Engineering (GPCE'09)*. ACM press, 2009, pp. 127–136.

[32] OSCI, "SystemC," http://www.systemc.org.

[33] M. Zeller, G. Weiss, D. Eilers, and R. Knorr, "Co-Simulation of Self-Adaptive Automotive Embedded Systems," *8th IEEE/IFIP International Conference on Embedded and Ubiquitous Computing (EUC)*, Dec. 2010.