Model-Based Development of Middleware for Self-Configurable Embedded Real-Time Systems: Experiences from the DySCAS Project

Tahir Naseer Qureshi, Magnus Persson, DeJiu Chen, Martin Törngren, Lei Feng
Department of Machine Design, The Royal Institute of Technology (KTH), Stockholm, Sweden
{tnqu,magnper,chen,martin,leifeng}@md.kth.se

***Abstract*—This paper presents experiences from the model-based development of a framework for a middleware targeting the needs for self-management and context awareness in automotive systems. The major focus is on a simulation platform and a reference implementation of the middleware architecture. We also discuss challenges and possible future extensions.**

I. Introduction

Distributed computer systems are becoming ever more prevalent in everyday life. This is also true for automobiles. A modern car has several dozen embedded processors and multiple networks. The increased usage of electronics and software puts more emphasis on better development methods. Additionally, the trend in the automotive industry is towards standardized software development.

AUTOSAR [1] is an evolving automotive approach to flexibility and development efficiency. It has its focus on configuration management, application and platform standardization while also intending to provide extended support for quality assessment(e.g. [2]).

Due to the needs of customization, enhanced maintenance, life-cycle support, optimized resource utilization and error handling advantages can be gained by considering dynamic (re)configuration of e.g. the run-time allocation of tasks to the processors on a network. and post-development software updates. This is especially interesting for systems such as automotive systems, which have comparatively longer life time compared to systems e.g. mobile phones and navigation systems.

A. DySCAS

DySCAS[1] [3], a research project funded by the European Commission, addressed the future needs for self-configuration, self-healing, self-optimization and self-protection, collectively known as self-management, in automotive systems. The major outcome of the project is the specifications of a framework for self-managing middleware [4]. It includes a core model specified in UML [5] and supporting activities [6], including formal verification, simulation [7], safety analysis [8], reference implementations [9] and demonstrators [10].

[1]Dynamically Self-Configuring Automotive System

One of the most interesting features of DySCAS is the multidisciplinary integration of technologies such as control systems, autonomic computing, middleware and policy-based computing. In addition, it also provides a ground for research on different mechanisms for resource management, quality of service, autonomic configuration etc.

The following sections give a brief overview of the DySCAS architecture, modeling, simulation and DyLite [11], one of the reference implementations. We conclude with a discussion on experiences and possibilities for future enhancements.

II. Architecture overview

DySCAS provides a systematic approach to dynamic architecture through middleware. The advantages are efficient and reliable field maintenance and upgrades, integration of resources for information and functionality sharing and post-deployment time optimization according to environment conditions and internal status.

The specifications of the DySCAS reference architecture is organized into following three packages:

- *Information model* for specifying system configurability and variability, embedding the meta-information for run-time reasoning and decision-making, capturing the environmental conditions and system internal status and describing the expected dynamic configuration tasks and activities.
- *Component model* for packaging configuration management data and control functions, exploiting a policy-based mechanism for the implementation of reasoning and decision algorithms and defining the fundamental MW behaviors in regards to execution, interaction and synchronization.
- *Architecture model* comprising of specifications of a *data hierarchy* and *layered control strategy* as well as specification of the structuring and deployment of middleware services that together perform distributed configuration management with activities like context data consolidation, variability resolution, change negotiation and commitment.

The DySCAS middleware services which provide support for self-contained overall all configuration management are themselves only offline configurable. As the major focus

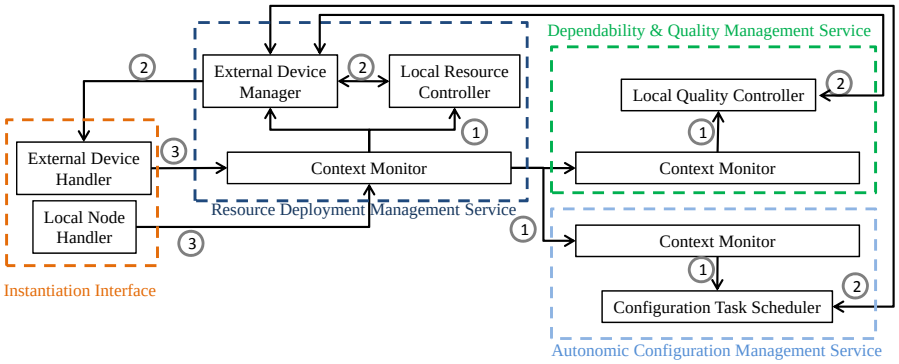

Fig. 1. Service interactions for attachment of a new device: (1) Context dissemination, (2) Service requests/feedbacks, (3) Platform / Device status.

is on the information and functional complexity, DySCAS provides a well-delimited system dynamic adaptability and configurability. For detailed specifications, readers are referred to [4].

III. Model transformation and Simulations in DySCAS

Verification and validation of a design in its early phases is beneficial in terms of identifying possible shortcomings and errors and providing design feedback. For embedded systems, this can be achieved by simulation and/or small scale prototype implementation. It is rather common that there is a gap between "design" or constructive models and analysis models. Compare for example between Simulink and UML models.

One of the major challenges of DySCAS was the need of using several formalisms and models for describing the system structures, behaviors of various types such as computation and communication. For example, a decision function can be modeled as logic expressions, mathematical expressions [12], policies [4], state machines and control flow [7]. One particular transformation is between UML and Simulink for the purpose of early architecture analysis, verification and validation. Table I gives an overview of the transformations. Further details can be found in [13].

TABLE I
Transformation scheme between the architecture design model in UML and Matlab/Simulink/SimEvents

UML	SimEvents/Simulink
Component model structure	Subsystem
Ports	SimEvents Connection port
External Signals	Entities
Signal attributes	Attributes
Decision functions	Attribute function
Execution controller	Stateflow
Computational modules	Combination of gates, queues and servers
Activity diagrams	Combination of Stateflow, gates, queues and servers

A. Base simulations with SimEvents

In DySCAS, a complete library has been developed for the DySCAS core services. The services in the library are limited to the component ports, execution controllers and input queues. The context monitor and the decisions functions are left open for the users. A few of the DySCAS scenarios were also simulated [7]. Figure 1[2] shows an overview of interactions between different DySCAS services for the scenario of the attachment of a new device to the vehicular network. Based on available context information such as available resources, security authentication etc., the middleware was able to accept/reject the new devices.

B. Detailed simulations with TrueTime

An elaborated simulation of DySCAS was carried out using the TrueTime toolbox for simulation of distributed control systems [14]. TrueTime provides simulation support for a set of platform features such as task and network scheduling etc.

Two major simulation studies have been carried out, focusing on the design solutions for resource optimization, quality of service (QoS), and device/service discovery and integration [15].

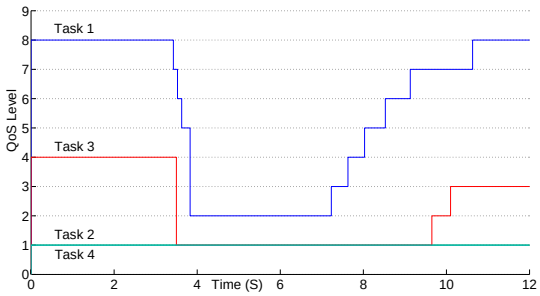

Fig. 2. Change of QoS Levels of application tasks

Figure 2 shows an output from a simulation run [4]. The scenario includes four application tasks, capable of running in different QoS modes. Each task is specified by its resource consumption, benefit and significance levels. The middleware decisions are based on a suite of algorithms to maximize the system benefit by allocating more resources to more important

[2]The numbers in the figure only represent the three types of signals. For details refer to [7]

tasks. These algorithms provide not only an approximate solution based on resource estimates, but also reject unknown disturbances using a feedback mechanism.

IV. Implementing DySCAS on hardware: DyLite

The development of DyLite has been carried out using traditional development platforms, according to the DySCAS architecture specified in UML. The transformation has been done manually due to a tool limitation in regards to code generation from UML.

A. Hardware Platform

DyLite was developed on a networked platform with two types of ECU:s (Electronic Control Units): the *Movimento Puma* and the *Freescale MCF5213EVB evaluation boards*. Here, the powerful Movimento Puma has been used as a *master* node, with the role of providing a name service, algorithms for configuration resolution, and global synchronization of the control actions performed by local middleware services on each node. All nodes are connected by a CAN network.

B. Task Model

The implemented task model in DyLite has several specific properties. First, a task may have more than one *QoS mode/ region*, each using different amounts of resources from the platform. The applications can switch between these modes using a function call `switch_region()` of the DyLite API. When this function is called, the middleware returns the next mode the application should use. The middleware may also block the application indefinitely upon calling the function (e.g. if overload occurs), if this is acceptable according to the application requirements.

To help the middleware control the resource utilization, and to ensure that e.g. hard deadlines will be met, each application comes with a specification of its meta-data, describing e.g. its functional dependencies and expected resource budgets (CPU time, communication and memory needs). The specification can also contain *utility* values to be used in the case of conflicts over resources between applications. In Dylite such meta-information are described in terms of *delivery notes*. Since the delivery notes give hard limits on the resource utilization, they can be used to form *contracts* between the applications and target platforms. This means that every application will be guaranteed the requested resources provided that the application does not exceed the agreed resource usage.

C. Communication Protocol

In figure 3, the communication protocol of DyLite are displayed. Both broadcast communication from a server to several subscribers, and private communication between the server and a single client are supported. A communication API containing function calls available to the applications for interaction with the middleware is also illustrated.

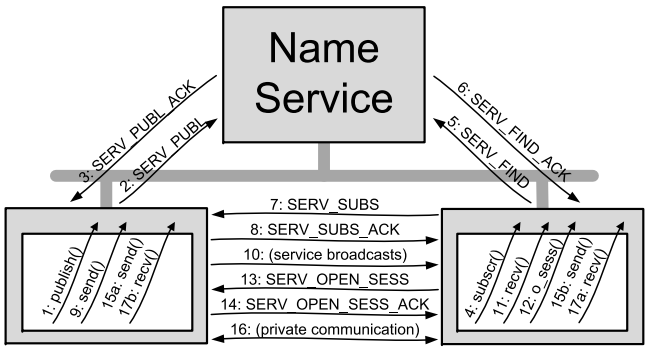

Fig. 3. The sequence of function calls and messages passed over the networks.

D. Reconfiguration

The *configuration problem* appears when new applications are added or removed, . Optimal allocation of applications to the nodes is an NP-complete problem. In DyLite, a simplified algorithm based on heuristics has been used. The algorithm, illustrated in figure 4, is a minor variation of the algorithm presented in [12].

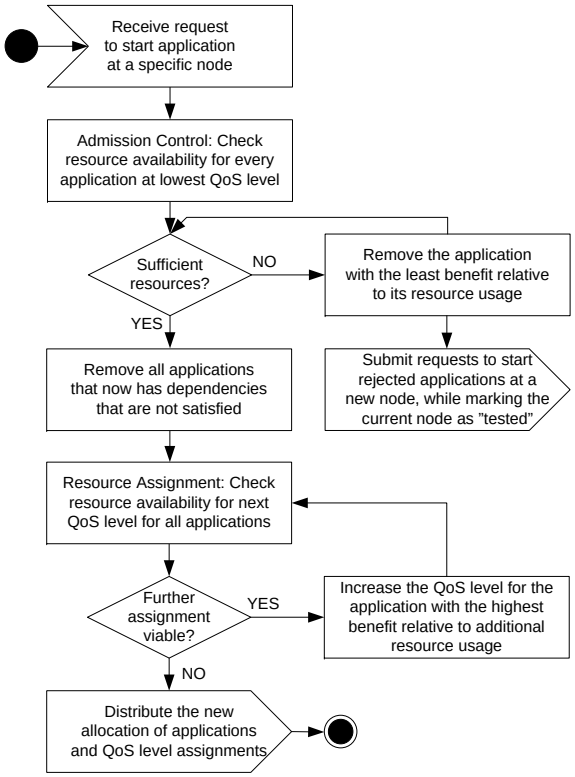

Fig. 4. The used reconfiguration algorithm, which is triggered when e.g. a request to start a new task is received. Please note that *all* applications are considered, not only new ones, as an old task may be less important than a new one and hence should be stopped.

If applications and communication links have appropriately set scheduling properties (e.g. priorities if priority-based scheduling is used), simple schedulability checks within the

admission control part of the reconfiguration algorithm can be used to ensure the applications' contracts.

V. Experiences summary

The concept of self-configuring embedded systems is new. Many of the types of modeling and analysis features are provided, however by tools which are not integrated. The implication of this is that of the general MDE problem, lacking systematic approaches for model transformations and synthesis. The aspect of variable structures is often not supported (for example not by UML). The transformation from architecture description in UML to functional design in SimEvents was carried out manually and can be considered as the first step towards an automated transformation. There is still a lot to be done in terms of tool development / integration and model transformations.

The development of DyLite has also pointed out several limitations with a traditional development approach. Deriving timing properties (e.g. WCET) through measurement and/or formal timing analysis takes a lot of time. Furthermore, it would be done automatically when the application code changes. This also includes automatic generation of the meta-data (delivery notes) based on a formal system architecture model.

VI. Conclusions

In this paper, we have presented our work on modeling, simulation and implementation of a dynamically reconfigurable middleware framework, DySCAS. During the work, several development challenges have been encountered and it is clear that the present development methodologies are not sufficient to reliably handle the complexities present in real-world systems with a DySCAS approach; better tool support and integration is needed. Here, model-based engineering forms a promising initiative.

References

[1] Autosar Consortium. [Online]. Available: http://www.autosar.org

[2] TIMMO Project. [Online]. Available: http://www.timmo.org

[3] DySCAS Consortium. [Online]. Available: http://www.dyscas.org

[4] DySCAS Consortium, "D2.3 DySCAS system specification (final drop)," 2009, project deliverable. [Online]. Available: http://www.dyscas.org/downloads.htm

[5] Unified Modeling Language. [Online]. Available: http://www.uml.org

[6] DySCAS Consortium, "D4.3 evaluation report," 2009, project deliverable. [Online]. Available: http://www.dyscas.org/doc/DySCAS_D4.3.pdf

[7] T. Naseer Qureshi, D. Chen, M. Törngren, L. Feng, and M. Persson, "Experiences in simulating a dynamically self-configuring middleware: A case study of DySCAS," Mechatronics Lab, Department of Machine Design, Royal Institute of Technology (KTH), Stockholm, Sweden, Tech. Rep. TRITA-MMK 2009:04, ISSN 1400-1179, ISRN/KTH/MMK/R-08/01-SE, 2009.

[8] L. Feng, M. Törngren, and D. Chen, "Safety analysis of dynamically self-configuring automotive systems," Mechatronics Lab, Department of Machine Design, Royal Institute of Technology (KTH), Stockholm, Sweden, Tech. Rep. TRITA-MMK 2008:13, ISSN 1400-1179, ISRN/KTH/MMK/R-07/12-SE, 2008:13.

[9] DySCAS Consortium, "D3.1 DySCAS specification of reference implementation and validation applications," 2009, project deliverable. [Online]. Available: http://www.dyscas.org/doc/DySCAS_D3.1.pdf

[10] DySCAS Consortium, "D3.3 DySCAS demonstrator application and specification," 2009, project deliverable. [Online]. Available: http://www.dyscas.org/doc/DySCAS_D3.3.pdf

[11] M. Persson, J. García, L. Feng, D. Chen, T. Naseer Qureshi, and M. Törngren, "Dylite: Design, implementation and experiences," Mechatronics Lab, Department of Machine Design, Royal Institute of Technology (KTH), Stockholm, Sweden, Tech. Rep. TRITA-MMK 2009:06, ISSN 1400-1179, ISRN/KTH/MMK/R-08/01-SE, 2009.

[12] L. Feng, D. Chen, and M. Törngren, "Self configuration of dependent tasks for dynamically reconfigurable automotive embedded systems," in *Proceedings of 47th IEEE Conference on Decision and Control*, Cancún, Mexico, Dec. 9 – 11 2008.

[13] T. Naseer Qureshi, D. Chen, M. Törngren, L. Feng, and M. Persson, "On mapping UML models to Simulink/SimEvents: A case study of a dynamically reconfigurable middleware," Mechatronics Lab, Department of Machine Design, Royal Institute of Technology (KTH), Stockholm, Sweden, Tech. Rep. TRITA-MMK 2009:**XX**, ISSN 1400-1179, ISRN/KTH/MMK/R-08/01-SE, 2009.

[14] TrueTime. [Online]. Available: http://www.control.lth.se/truetime

[15] L. Feng, D. Chen, M. Persson, T. Naseer Qureshi, and M. Törngren, "Dynamic configuration and quality of service in autonomic embedded systems," Mechatronics Lab, Department of Machine Design, Royal Institute of Technology (KTH), Stockholm, Sweden, Tech. Rep. TRITA-MMK 2008:12, ISSN 1400-1179, ISRN/KTH/MMK/R-07/12-SE, 2008.