CHROMOSOME: A Run-Time Environment for Plug & Play-Capable Embedded Real-Time Systems

Christian Buckl, Michael Geisinger, Dhiraj Gulati, Fran J. Ruiz-Bertol
fortiss GmbH
Cyber-Physical Systems
Munich, Germany
Email: {buckl, geisinger, gulati, ruiz}@fortiss.org

Alois Knoll
Technische Universität München
Department of Computer Science
Robotics & Embedded Systems
Email: knoll@in.tum.de

***Abstract*—Developers of embedded systems are increasingly facing requirements concerning adaptivity. The term adaptivity covers several different aspects. In this paper, we present an innovative open-source run-time system that enables adding and removing applications and adapting to changes in the hardware topology while still guaranteeing traditional requirements of embedded systems such as real-time or safety. The system uses a data-centric approach to achieve a modular application design and to enable the interaction of application components created by independent developers. Using a requirements-centric approach, the system can reserve the resources such as processor time, memory or network bandwidth required by application components. New applications can only be added if enough resources are available. The paper details these concepts and presents the architecture of the run-time system.**

I. Introduction

Adaptivity is becoming an important requirement for embedded systems. While in the past embedded systems were designed once and then remained unchanged until the end of their life cycle several years later, modern embedded systems and especially their software need to be adapted frequently. Adaptivity might be used to achieve design goals such as system integration, system dynamics and reflection.

System integration: In the past, embedded systems were often isolated and typically implemented one unique function, e.g., a control task. Due to performance advances of modern embedded hardware and network technologies, the resulting systems have become increasingly complex. As a result, embedded systems today implement several features in a distributed system. A prominent example is automotive software running on up to one hundred electronic control units inside a single car and executing thousands of different functions ranging from driving to infotainment. However, these functions and domains have different life cycles and customers are increasingly demanding the ability to update or even add new functions [1], [2].

System dynamics: One more reason to require adaptivity is the dynamics of embedded systems. One example are sensor networks where nodes might appear or vanish. In this case, the configuration should adapt to the underlying topology to deliver the best possible performance. A similar requirement can be derived for fault-tolerant systems. In the past, reconfiguration was designed manually. Due to the rising complexity and accordingly the increasing solution space, an automated solution is desirable.

Reflection: Another kind of adaptivity is based on reflection. If the run-time system is capable of understanding and interpreting the services that it provides, it can adapt its functionality to deliver the optimal performance. For instance, the system may analyze the data flow in a distributed application and route the traffic over redundant communication channels in order to optimize bandwidth usage or safety.

In this paper, we focus on the first two aspects of adaptivity: changes of the application induced by the user and adaption due to topology changes. The idea is to provide an appropriate run-time environment for embedded real-time systems that supports plug & play regarding both the software and the network level. Although many run-time environments and middleware solutions are available, they do not match the requirements of adaptive embedded systems. On the one side, run-time systems known in the embedded domain typically rely on a *static configuration* to satisfy requirements such as real-time behavior or safety. On the other side, middleware solutions from the web service domain satisfy *adaptivity requirements*, but do not guarantee hard real-time capability or fulfillment of resource constraints. The paper addresses the challenge of fulfilling adaptivity needs while still guaranteeing traditional requirements from the embedded systems domain. For instance, changes of the configuration must not result in the violation of real-time requirements or resource conflicts.

As a main contribution, the paper presents an approach for combining these two conflicting requirements. The general idea is to use a requirements-centric approach in contrast to the usually applied configuration-centric approach of today's embedded run-time environments. In addition, we replace the commonly used message-centric approach by a data-centric approach with preceding domain modeling, i.e., we specify communication requirements based on the data itself. The approach is implemented in the open-source run-time environment CHROMOSOME[1] (abbreviated as XME).

The paper starts with a discussion of the basic concepts in Section II. Section III presents the architecture of the developed run-time environment. A tool for specification of application requirements and the according tailoring of the run-time environment is presented in Section IV. An overview of related research projects and a discussion of related work is found in Section V. Section VI summarizes the paper and gives an outlook of future work.

[1] http://chromosome.fortiss.org/

II. Basic Concepts

This paper presents a cross-platform run-time environment (RTE) called XME that provides adaptivity features to the application layer. Modifications in the application layer may be triggered either by the end user or by changes in the system's topology, i.e., nodes may appear or disappear at run-time. We assume that changes occur rarely compared to the regular execution of the system. Following the plug & play terminology, we therefore focus on an efficient *play-phase*, while we assume that efficiency in the *plug-phase* is not of great importance. Scheduling, network routing and other decisions are based on *configuration/lookup tables*. Reservation of required resources happens already in the plug-phase. This guarantees a very efficient execution in the play-phase in contrast to plug & play capable systems that dynamically calculate every decision. The drawback is that this reservation strategy requires a pessimistic system design in which more resources (e.g., CPU time, memory) have to be available than initially required.

In the plug-phase, XME calculates the required configuration changes as a shadow configuration in parallel to executing the existing configuration. Therefore, the normal system execution is not interrupted. XME updates the lookup tables after finding a valid new configuration. Furthermore, in order to support real-time reactions to expected changes, an adequate shadow configuration may be calculated in advance. This is especially important to achieve fault-tolerance in real-time systems. The time for system reaction is then limited to the duration of detecting and verifying a component's failure as well as selecting the appropriate shadow configuration. This is similar to today's manually implemented error reactions.

The main concept to support the calculation of correct configurations in the presence of extra-functional requirements (e.g., timing guarantees) is a *requirements-centric* approach. Instead of manual configuration of the RTE with respect to the application components, system developers directly specify respective requirements. This can be real-time requirements, such as end-to-end latency or jitter (as discussed in [3]), or requirements regarding the safety level if the RTE provides appropriate mechanisms.

Another key requirement to achieve plug & play is the existence of a mechanism to integrate new software components. Popular mechanisms to achieve this requirement include service oriented design and *data-centric design*. Service oriented design is based on request/response communication and is used for example in web services. The data-centric approach, suggested for example by OMG DDS [4], focuses on specification of communication requirements instead of defining fixed communication relationships. Since embedded systems very often communicate in a unidirectional way to save resources, we implemented XME based on the data-centric approach. Request/response style communication is implemented on top of the data-centric framework.

The data-centric approach is based on publish-subscribe semantics. In a publish-subscribe environment, components declare the types of data they produce and consume, commonly referred to as topics [5]. XME sets up a chain of application components communicating with each other based on this *functional* interface description.

To achieve interoperability between components of different developers that may even be unaware of each other, it is required to agree upon common topics. We achieve this requirement by a *domain modeling* approach in which so-called *dictionaries* predefine the available topics. This approach ensures that syntax and semantics of exchanged data are unambiguously defined. Apart from specifying the semantic types of data items, the dictionary also provides *attributes* to describe data items in a more detailed way. Attributes denote, for example, the possible range of a data value, the precision or the safety-criticality level. The matching between subscribers and their potential publishers accounts for attributes as well.

Finally, modularity requirements also affect XME. Embedded systems very often consist of heterogeneous computing platforms with very different resources and requirements. This fact must also be taken into account when designing an RTE. Already in 2006, Buttazzo stated the requirement that:

> "A true component-based approach should separate mechanisms from policies in order to replace a scheduling algorithm or a resource management protocol without affecting the applications and the others components." [6]

As a result, XME is designed to be very modular. We distinguish between different capabilities of XME nodes. Some resource-constrained nodes might not need to support plug & play in the sense of installing new software, but may only need to support its integration into a network of other nodes. Instead, more powerful nodes might be able to calculate the system's topology and decide on reconfiguration actions. Therefore, the system developer specifies which kind of services is added to the RTE instance of a specific node. Furthermore, different implementations of one RTE component might be available. As an example, the scheduler might be available as a time-triggered, event-triggered or hybrid scheduler. The developer or the tooling can then select the most appropriate implementation.

III. RTE Architecture

The XME RTE abstracts from resources such as processor, memory and communication, and provides a uniform communication mechanism to the application components running on top of the RTE. We call the federation of computational nodes running XME with the ability to communicate with each other an XME *ecosystem*. Currently, we assume that an ecosystem has one *master node*, which orchestrates the login and plug & play process. Communication between the application components executed on the different nodes is performed in a directed fashion, i.e., the publishing node sends the data directly to the subscribing node(s).

XME assumes that the application itself is composed of modular and reusable components. The application components communicate via publications and subscriptions with each other. Other interaction mechanisms are excluded due to safety reasons. The architecture of the XME RTE is depicted in Figure 1 and described in detail in the following.

A. Components for RTE Play

The three main components that are required to execute applications and to implement the data-centric communica-

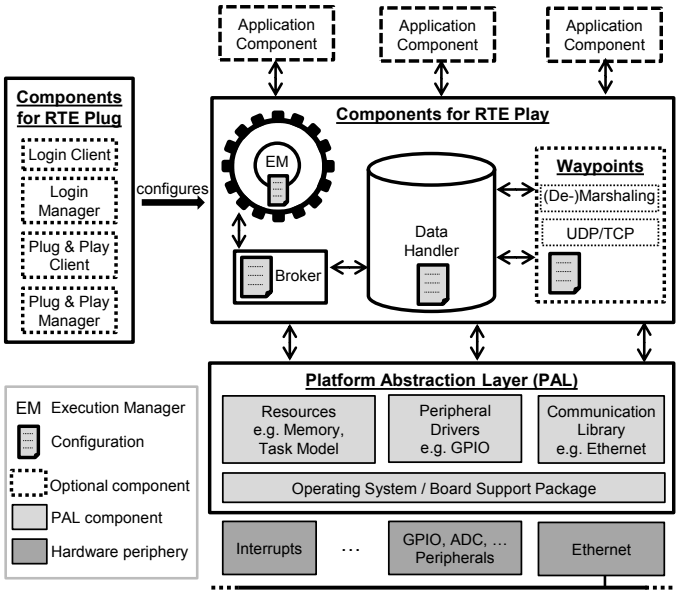

Fig. 1. Architecture of the XME run-time environment.

tion (DCC) on a node are the *Data Handler*, the *Broker* and the *Execution Manager*. The Data Handler provides an API for each application component to read its subscribed data and to write data that it publishes to other components. The Broker's configuration lists the data subscription requirements of each application component. It monitors the availability of the respective data in the Data Handler and enables the components for execution in the Execution Manager as soon as all requirements are satisfied. The Execution Manager is responsible for executing components enabled by the Broker. In addition, it monitors the execution of components with respect to their specified worst-case execution time (WCET) and reports violations. The configuration of these main components is determined by lookup tables as mentioned in Section II.

B. Waypoints

Depending on the network topology, the RTE performs several actions to transfer the data. If publisher and subscriber reside on two different nodes, a message is sent over the network. This includes converting the data to a common byte order (marshaling). However, further actions might also be necessary, such as adding a CRC checksum to detect message corruption or sending of data in a redundant way in safety-critical scenarios. One solution to solve this issue would be the introduction of according network stack layers. However, in embedded systems, the requirements on the processing of data during send and receive are very heterogeneous and a layering approach might result in a huge overhead. Furthermore, some processing steps are the same for different communication protocols. Therefore, we introduce a modular concept called *waypoints*. Waypoints are lightweight software components that implement a concrete mechanism, e.g., marshaling. Multiple waypoints can be configured to form a data processing chain. This leads to a lightweight yet powerful design to manage heterogeneous requirements. XME applies a set of predefined rules in order to select the waypoints to be included in a specific processing chain during the plug-phase.

C. Platform Abstraction Layer (PAL)

XME offers a platform abstraction layer (PAL) which consists of a uniform API to access basic services provided by the underlying operating system. Services covered on all platforms include memory management, a task model and communication mechanisms (compare Figure 1). To maintain platform independence, software components should access low-level features through the PAL. Currently XME supports several operating systems, but we also intend to provide a version that runs without operating system support.

D. Components for RTE Plug

To cope with changes in the XME ecosystem, we provide several components for the plug-phase, namely login-related components and plug & play-related components. Adaptivity features are only available on a node if those components are present. When an external node logs into (i.e., joins) an XME ecosystem, the *Login Client* on that node establishes an initial communication route to an existing *Login Manager*, which subsequently enables plug & play and application components to use DCC. The *Plug and Play Client* of a logged-in node is responsible for announcing the "pluggable" application components on this node to the *Plug and Play Manager*. Information exchange happens via *manifests* that describe the publications, subscriptions and requirements of the respective components. The Plug and Play Manager collects this information and sends it to the *Logical Route Manager* (not shown in Figure 1), which calculates all possible communication routes. Subsequently, the Plug and Play Manager checks the feasibility of the calculated routes in cooperation with the affected Plug and Play Clients by invoking various configurators. These configurators include the global *Network Configuration Calculator* (not shown in Figure 1) to check whether enough bandwidth is available, but also configurators on the affected nodes to guarantee the required resources. Only if reservation of required resources is successful, the Plug and Play Manager initiates the switch to the new configuration. Otherwise, XME rejects the topology change.

IV. Tooling

The complexity of distributed embedded systems forces developers to rigorously define the requirements for those systems. Specialized tools help to map the requirements to the implementation level. CHROMOSOME Modeling Tool (XMT) is an Eclipse-based model-driven design tool, which defines an iterative workflow for specification, implementation and evolution of distributed embedded systems in various domains such as industrial automation, automotive and robotics. The specified requirements include communication patterns, real-time constraints and sanity criteria and are traceable to the different steps of the workflow. XMT allows declaring topic dictionaries, to develop application components and to specify the structure of the XME ecosystem. The XMT model supports validation in order to detect design problems early in the development phase. Finally, we apply code generation to configure the RTE. This concept also allows to generate a completely static (i.e., off-line generated) RTE configuration that has no run-time adaptivity, but also no run-time overhead.

V. Application Areas and Related Work

A. Related Research Projects

The development of XME is influenced by other research projects from different domains. XME combines those requirements and generalizes them in order to provide a domain independent RTE. Typical requirements include run-time adaptivity and time-triggered behavior with real-time guarantees.

The RACE project [7] aims at reducing automotive architecture complexity while enabling new adaptivity features. XME provides RACE with a run-time environment that enables plug & play while giving guarantees on the execution of applications. For this purpose, the Execution Manager is configured for hard real-time scheduling based on WCET of components. RACE requires several new features such as spatial partitioning of application components and a safety manager with respective state management. Spatial partitioning is realized with PikeOS as underlying operating system. XME runs in a special "master" partition. Addition and removal of software components is realized by starting/stopping them in their respective partitions. Hence, applications can be plugged at run-time without affecting other parts of the system.

In contrast, AutoPnP [8] focuses on providing plug & play functionality to the industrial automation domain. Today's individualization of products makes it necessary to reconfigure production facilities on demand without stopping the production process. XME realizes the software aspect of this reconfiguration. AutoPnP uses an event-triggered "best effort" execution model with known event arrival rates. Hence, the Execution Manager is configured to ignore WCET of components. Reconfiguration includes the automatic detection of hardware components and the according activation of software components. A high-level control monitors and adapts the data flow on demand. In order to exchange information with existing systems, XME also interfaces with ROS [9] and PLCs to fit the setting of AutoPnP.

B. Related Work

XME borrows the concepts of data-centric communication from OMG DDS [4]. Unlike in DDS, in XME data is delivered only to the nodes that need the information. This optimizes the communication bandwidth and safety and security requirements are more easily addressed. A subset of the DDS quality of service constraints is represented in XME in form of explicit requirements in XMT. In addition, XME provides the attribute concept for fine-grained publisher/subscriber matching.

The Robot Operating System (ROS) [9] is a well-known middleware from the robotics domain. Although ROS and XME share some design concepts, XME provides more guarantees with respect to the behavior of the distributed application. This is due to the formal specification and requirements analysis approach in XMT.

FRESCOR [10] is a contract-based framework for real-time embedded systems, where applications specify their requirements as contracts. The contracts of all applications are negotiated at run-time. Contracts are similar to XME manifests and negotiation corresponds to the tasks of the XME Plug and Play Manager. However, there is no way to statically generate a configuration from the contracts as it is possible with XMT.

VI. Summary and Outlook

Although XME shares a number of concepts with existing run-time environments and middlewares, such as platform abstraction and communication, it focuses primarily on run-time adaptivity. For this purpose, explicit requirements are specified in model-driven tooling that is used at run-time to assess the possibility of reconfiguration. Dedicated software components that are transparent to application components implement adaptivity concepts such as plug & play and login/logout of nodes. In order to efficiently run the applications, the execution of crucial software components is based on lookup/configuration tables. Both XME and XMT are available under the Apache License, version 2.0.

The future roadmap of XME includes integration of various new features: we currently support real-time guarantees only regarding a single node, but not the related communication. In the future, we also want to consider end-to-end latency requirements. Second, the plug & play capability is currently restricted to adding or removing complete nodes. In the next versions, we will enable deployment of new application components as software binaries. Third, we want to increase the comfort of developing embedded systems by adding new features that help the developers to guarantee extra-functional requirements: health monitoring will be used to continuously monitor the nodes in an XME ecosystem and adapt the configuration automatically. Security aspects will be integrated to restrict the topics software components can subscribe to. Finally, we will extend our tooling to not only support the development and deployment phases, but to provide interesting features at system run-time as well, such as monitoring and visualization.

References

[1] C. Buckl, A. Camek, G. Kainz, C. Simon, L. Mercep, H. Stähle, and A. Knoll, "The software car: building ICT architectures for future electric vehicles," in *Proceedings of the 2012 IEEE International Electric Vehicle Conference (IEVC 2012)*. IEEE, 2012, pp. 1–8.

[2] M. Di Natale and A. Sangiovanni-Vincentelli, "Moving from federated to integrated architectures in automotive: The role of standards, methods and tools," *Proc. of the IEEE*, vol. 98, no. 4, pp. 603–620, Apr. 2010.

[3] C. Buckl, I. Gaponova, M. Geisinger, A. Knoll, and E. A. Lee, "Model-based specification of timing requirements," in *Proc. 10th ACM Intl. Conference on Embedded Software*. ACM, Oct. 2010, pp. 239–248.

[4] G. Pardo-Castellote, "OMG data-distribution service: Architectural overview," in *Proceedings of the 23rd International Conference on Distributed Computing Systems*. IEEE, May 2003, pp. 200–206.

[5] G. Pardo-Castellote, B. Farabaugh, and W. Rick, "An introduction to DDS and Data-Centric Communications," RTI, Aug. 2005. [Online]. Available: http://www.omg.org/news/whitepapers/Intro_To_DDS.pdf

[6] G. Buttazzo, "Research trends in real-time computing for embedded systems," *SIGBED Rev.*, vol. 3, no. 3, pp. 1–10, Jul. 2006.

[7] S. Sommer, A. Camek, K. Becker, C. Buckl, A. Zirkler, L. Fiege, M. Armbruster, G. Spiegelberg, and A. Knoll, "RACE: A centralized platform computer based architecture for automotive applications," in *Proc. of the 2013 Vehicular Electronics Conference and the International Electric Vehicle Conference (VEC/IEVC 2013)*. IEEE, Oct. 2013.

[8] N. Keddis, G. Kainz, C. Buckl, and A. Knoll, "Towards adaptable manufacturing systems," in *IEEE International Conference on Industrial Technology (ICIT 2013)*. IEEE, Feb. 2013, pp. 1410–1415.

[9] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: an open-source robot operating system," in *Proc. International Conference on Robotics and Automation*, 2009.

[10] FRESCOR project, "Framework for Real-time Embedded Systems based on COntRacts," 2008. [Online]. Available: http://www.frescor.org/