Towards Safe Dynamic Updates of Distributed Embedded Applications in Factory Automation

Kilian Telschig
Corporate Technology
Siemens AG, Munich, Germany
kilian.telschig@siemens.com

Alexander Knapp
Institute for Software and Systems Engineering
University of Augsburg, Germany
knapp@informatik.uni-augsburg.de

***Abstract*—In future production systems the speed of adaptation to changing market needs becomes increasingly important. As automation processes are carried out by flexible, software-defined machines like robots, adaptivity could be achieved largely through software updates. For technical and economic reasons it would be an advantage to update the automation system without stopping production. The challenge is to enable consistent software updates to running distributed embedded applications while keeping the timing requirements of messages and tasks. We describe the kinds of updates to be supported and give a brief first outline of our technical solution for this problem: A real-time container infrastructure that runs embedded components and is able to reconfigure the running distributed application due to a reconfiguration plan to be designed by the plant operator and to be verified by the plant engineering system.**

I. Introduction

In the age of digitalization keeping pace with the market is more important than ever before. But in factory automation, fast adaptation is difficult, because factories are built for very dedicated purposes while engineering and commissioning can take months and years. Therefore, changes to such a special purpose plant are expensive, slow, and thus often unprofitable. Any improvement on this leads to significant competitive advantages and therefore the adaptation speed is becoming a critical success factor for manufacturers.

Fortunately, production processes are increasingly carried out by software-intensive systems controlling multi-purpose machines like robots, 3D printers and CNC milling machines. Such systems can potentially be adapted instantly by updates that use the equipment in a different way. Moreover, future automation systems will need to adapt constantly, whereas a full shutdown of the overall plant would be uneconomical and sometimes even technically impossible. Therefore, it must be possible to roll out updates during production.

On the other hand, updating a running automation system is challenging. As the software is embedded in a technical process, it consists of tasks and signals with real-time requirements that need to be assured by the runtime environment to avoid hazards. Additionally, the automation application is distributed over multiple controllers. As synchronization between nodes takes longer than common timing deadlines of tasks there is no global state in which an update would be safe (except when stopping production).

Thus, the challenge is to update distributed embedded applications at runtime with at least the following guarantees:

- States and messages will not get lost.
- Real-time requirements of all continued tasks and all valid messages will be met.
- The distributed system will be consistent at any instant.

Our vision is a real-time container technology, inspired by modern software engineering practice. Container technologies like Docker [1] and Linux Containers (LXC) [2] provide a dynamic and lightweight mechanism for software isolation and resource control, e.g. for continuous integration jobs or as app execution context. On top of that, we need to add reconfiguration mechanisms for dynamic updates of automation applications that ensure real-time requirements in the presence of distributed dependencies. As a side effect, real-time containers will provide additional benefits like enhanced security, reliability and compatibility.

In the rest of this paper, we proceed as follows. The next Sect. II refers to related future automation visions and approaches and describes the scope of our research activities. In Sect. III we outline our idea of real-time containers and related means supporting the reconfiguration. The final Sect. IV concludes this paper and gives an outlook of our planned future work towards safe factory updates.

II. Reconfiguration Needs of Future Automation Systems

We refer to visions and approaches related to dynamic reconfiguration of automation systems. Due to domain requirements and to the system complexity in automation speeding up adaptivity also leads to an increased demand for automatic assurance and enforcement of quality attributes such as real-time constraints, including consistency properties to be maintained during reconfiguration.

A. Future Automation Visions

A recent trend of automation technology is the ability to inspect the machine data of production systems via a cloud-based analytics platform. Usually, an additional gateway device needs to be placed into the factory that reads the data from the Programmable Logic Controllers (PLC), e.g. using

the OPC UA standard [3]. To cope with limited transfer rates and buffer sizes, but also with privacy, the gateway must preprocess the data still inside the plant (also referred to as edge-centric computing [4]). Depending on how critical a loss of data would be in the related domain, preprocessing tasks may have hard real-time requirements. When changing the analytics application in the cloud, the gateways' communication channels and the deployed preprocessing functionality will need to be adapted accordingly. While being reconfigured the gateway still must not lose any data. Therefore the ability to update real-time systems with quality guarantees would be beneficial for industrial analytics applications, even when not changing the production system itself.

Thinking one step ahead, the next requirement is then to optimize the plant according to the gained analytical insights [5], as also propagated by the DevOps movement [6]. Obviously it would be a great benefit to let the production system continue its job during the upgrade. This in turn increases the need for quality assurance and enforcement, especially if also third-party components from a digital marketplace are installed. Even more critical are visions towards an industrial Internet of Things (IoT) [7], where devices might reconfigure themselves to adapt to changing demand and equipment [8]. In this regards a dynamic reconfiguration mechanism with sufficient quality guarantees would be a step towards a centrally managed self-adaptive production system as favored by [9].

B. Required Reconfiguration Capabilities

To enable the previously described automation scenarios, distributed embedded applications will need to be component-based and support reconfiguration operations as described in this section. Overall, the embedded application is distributed over multiple nodes and consists of interconnected software components that communicate with each other potentially via network connections to control several machines in coordination. All of the basic interaction paradigms sender/receiver, client/server and publisher/subscriber between components shall be supported on the conceptual level closely following the Generic Component Model of MARTE [10].

Such an application is complex even without reconfiguration: The location and number of receivers, clients and subscribers for all interfaces must be respected in the communication configuration. Additionally, operation calls and event emissions trigger tasks in the corresponding components, influencing their resource needs. As client-server-communication is synchronous, the worst-case execution time (WCET) of a task calling an operation depends on the location and the WCET of the triggered task, which in turn might call operations.

Nevertheless, a plant operator shall be able to change such a system's configuration during its execution from a plant engineering system: the set of component instances, how they are connected with each other, how they are allocated to nodes and which resources they are given. For this purpose, future automation systems shall support changes to a running configuration consisting of the following reconfiguration operations:

1) Add Software Component to Running System
2) Move Running Software Component
3) Update Running Software Component
4) Remove Running Software Component
5) Update Running Software Composition
6) Add Equipment to Running System
7) Remove Equipment from Running System

To prevent damage to equipment, people and environment dynamic updates must be applied very carefully. On the one side, there is the impact on the technical process, which is out of the scope of this work. On the other side, the quality constraints of the distributed embedded application need to be kept. Ideally, the whole system shall be consistent before, during and after reconfigurations.

Our understanding of consistency is based on the ACID properties of transactions in database systems [11] adding the time dimension: Besides keeping all tasks' deadlines, the messaging constraints have to be ensured, even if a related component is reconfigured (for example consider relocation, update, ...). All components' states need to be preserved during the transition process. Additionally, fault detection and failure reaction mechanisms need to be preserved as the technical process continues. To make reconfiguration feasible in practice it must be possible to permit temporary quality degradation. Besides these exceptions at any instant either the old or the new configuration must be fully working.

C. Existing Approaches

In factory automation the standard IEC 61131-3 [12] defines programming languages for PLCs like Function Block Diagrams (FBDs). The standard IEC 61499-1 [13] extends this specification for distributed event-triggered FBDs. The standard is well established and can be used for component-based software engineering [14]. However, neither is reconfiguration of such a distributed FBD supported nor is client/server-communication. Apart from that no end-to-end quality contracts are used, which would be needed as input for quality assurance and enforcement approaches.

A promising related approach in factory automation is the recently proposed rtSOA [15]. It supports reconfiguration of distributed manufacturing systems and automatically resolves bus scheduling for inter-service communication with regard to WCETs of tasks. One central restriction of the approach is that it can only reconfigure the whole plant at once and when production has stopped as each service is directly associated with a production step. As a resulting architectural difference the inter-service-communication is unidirectional and the tasks associated with services are activated sequentially.

We think for future automation scenarios a holistic component-based software engineering approach as described by [16] needs to be adapted to dynamic distributed embedded applications. We are aware of approaches from various domains, e.g. the automotive standard AUTOSAR [17], which does not support reconfiguration. To the best of our knowledge no approach supports the capabilities described in II-B, e.g. relocation of stateful real-time components in conjunction with remote operation calls.

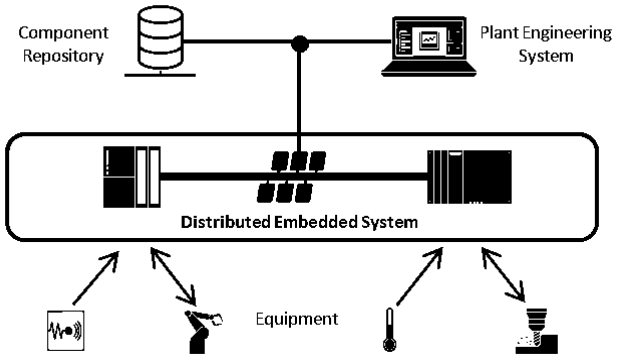

Fig. 1. An overview of the system architecture: The embedded application controlling the equipment is distributed over multiple nodes. The plant engineering system and the component repository are connected to each node to enable reconfiguration at runtime.

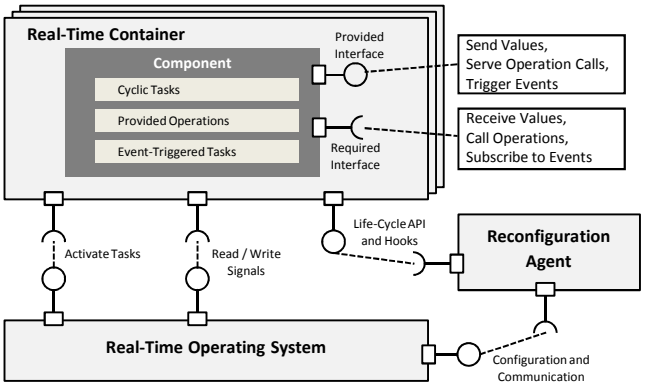

Fig. 2. Real-time containers: All instantiated components are executed in a dedicated container that ensures timely task execution and inter-component communication and supports life cycle operations for reconfiguration.

III. Reconfigurable Factory Vision

We describe our concept to make factory automation reconfigurable without downtime of the overall system. It consists of the following means (see Fig. 1 for a schematic overview of the system architecture):

Real-Time Containers as isolated execution context for embedded components providing resources as configured.

Reconfiguration Agents with privileges to execute reconfiguration instructions locally on a node.

The Plant Engineering System to design a working reconfiguration plan for the distributed embedded system.

A Component Repository inside the factory that provides component descriptions and binaries.

A. Real-Time Containers

A real-time container shall be an execution wrapper for a component that needs to be isolated from but at the same time connected to other components and resources, i.e., hardware access. The underlying architecture of the distributed embedded application shall be time-triggered as proposed by [18]. The container is responsible for the timely execution of a component's tasks and for the propagation and delivery of messages to and from other components' containers. Only via the container, components can interact with the runtime environment, with the hardware and with each other. The real-time container has to interact with the underlying RTOS to run the component accordingly (see Fig. 2):

- It requests execution of the component's tasks (translating event triggers to time triggers).
- It relays incoming and outgoing messages (signals, operation calls, events).
- It conveys system operations to provide the required hardware access.
- It watches the timing quality of tasks and messages and triggers failure reaction hooks of the application and the runtime environment.

For this purpose the real-time container needs a configuration compatible to the component's interface. The functional part of the interface declares the tasks that realize the functionality of the component and the required and provided interfaces for communication with other components and resource access (i.e. the ports). Additionally, the quality contract describes the timing behavior and constraints of a component's ports and tasks. However, the isolated component's interface alone is not sufficient, as its quality properties depend on remote communication and execution delays, too. Therefore the container configuration has to be created at the plant engineering system with the complete configuration of the distributed embedded application in mind (see Sect. III-C).

B. Reconfiguration Agent

The reconfiguration agent is a script execution engine running on each node. Its task is to receive and execute reconfiguration instructions from the plant engineering system as part of a global reconfiguration plan to be executed concurrently on all nodes while the embedded application is running. The reconfiguration agent shall provide the following reconfiguration capabilities:

Container Life-Cycle Management: Create/destroy, start/stop, snapshot/restore real-time containers.

Container Interception: Queue/forward specific messages, suppress tasks, inject/extract component state.

Runtime Environment Configuration: Create/delete bus signals, allocate/free container memory, modify task schedule.

Inter-Agent Coordination: Send/receive status messages from/to other agents.

Administration Connection: Receive reconfiguration scripts, download component binaries and container configurations.

Obviously, the procedure resulting from a reconfiguration plan has to be prepared and checked carefully at the plant engineering system before rolling it out.

C. Plant Engineering System

The plant engineering system helps the plant operator creating a safe reconfiguration plan. Its main task is to prevent

dangerous violations to the component contracts by checking each reconfiguration plan before rolling it out to the production system. This also includes checking the quality of service both of the new version of the distributed embedded application and the transition to this new version when executing the reconfiguration plan. For this check, the plant engineering system uses the component interfaces (including the timing regulations) and its knowledge about the network topology and the mapping of components to nodes, i.e. the configuration. An important note is that the effect of the reconfiguration to the technical process being controlled by the distributed embedded application is out of the scope of our concept and to be considered by the plant operator.

In the reconfiguration plan, the plant operator specifies a sequence of reconfiguration operations (e.g. move a component) using the reconfiguration instruction supported by the reconfiguration agent (see Sect. III-B). Besides sequential execution, he or she can also configure operations to run concurrently according to the atomicity principle, i.e. to take effect virtually instantaneously.

When the plant engineering system discovers that a given reconfiguration plan would lead to a non-feasible new system configuration, it shall deny the plan. If the resulting configuration is feasible, but the plant engineering system cannot find a reconfiguration procedure for which the integrity of the production system can be guaranteed during the update, then the plant operator either has to specify bounds of intermediate quality degradation to make the reconfiguration feasible, or to postpone the update to the next downtime of the factory. In no case the plant engineering system shall allow a plan that would violate the quality contract without explicit permission.

D. Component Repository

The component repository stores component binaries along with their interfaces. While the plant operator creates a reconfiguration plan, the plant engineering system uses the repository to check the existence of the required component versions and the compatibility of their interfaces. During reconfiguration, the reconfiguration agents download components from the component repository. Therefore it must be located inside the factory.

The plant operator can add and remove components to and from the repository, possibly via a central marketplace. The component repository ensures the integrity of the components for security (e.g. proof of origin), but also for quality (i.e. accordance of the component binaries to their interfaces). When using open-source components, it may also download and compile the code. For all components, the component repository shall also manage licenses and payment.

All in all the component repository is an important quality gateway for the production system.

IV. Future Work

We proposed a real-time container infrastructure aimed at facilitating safe reconfiguration of distributed embedded applications in factory automation. Real-time containers shall be an execution context for embedded components that on the one hand need to communicate with each other and with the equipment, but on the other hand need to have independent life cycles. Besides containers, a reconfiguration agent is running on each node of the distributed system. It does not control the technical process, but executes reconfiguration instructions according to a global reconfiguration plan designed beforehand by the plant operator at the plant engineering system. And finally, there shall be a component repository inside the factory that stores components and acts as quality gateway.

Our next step is a more systematic literature study to identify blind spots, including approaches from similar domains like automotive and avionics. In parallel we continue prototyping the real-time container infrastructure to get a better understanding of the problem and to evaluate the suitability of our architecture. Additionally, we are going to validate the approach in close cooperation with related projects inside Siemens using the prototype to implement a future automation scenario that requires dynamic reconfiguration (see Sect. II-A). Our hope is that one day the real-time container infrastructure will be a standard technology in factory automation.

References

[1] D. Merkel, "Docker: Lightweight linux containers for consistent development and deployment," *Linux J.*, vol. 2014, no. 239, 2014.

[2] linuxcontainers.org, "LXC," 2017, [Online; last visited May 8, 2017].

[3] S.-H. Leitner and W. Mahnke, "OPC UA–Service-oriented Architecture for Industrial Applications," *ABB Corporate Research Center*, 2006.

[4] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, "Edge-centric Computing: Vision and Challenges," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 5, pp. 37–42, Sep. 2015.

[5] J. S. Michels, "Industrial Connectivity und Industrial Analytics, Kernbausteine der Fabrik der Zukunft," in *Industrie 4.0 grenzenlos*, U. Sendler, Ed. Springer Berlin Heidelberg, 2016.

[6] O. Thomas, A. Varwig, F. Kammler, B. Zobel, and A. Fuchs, "DevOps: IT-Entwicklung im Industrie 4.0-Zeitalter," *HMD Praxis der Wirtschaftsinformatik*, vol. 54, no. 2, pp. 178–188, 2017.

[7] L. Da Xu, W. He, and S. Li, "Internet of Things in Industries: A Survey," *IEEE Trans. on Ind. Informatics*, vol. 10, no. 4, pp. 2233–2243, 2014.

[8] J. Lee, B. Bagheri, and H.-A. Kao, "A Cyber-physical Systems Architecture for Industry 4.0-based Manufacturing Systems," *Manufacturing Letters*, vol. 3, pp. 18–23, 2015.

[9] N. Kaur, C. S. McLeod, A. Jain, R. Harrison, B. Ahmad, A. W. Colombo, and J. Delsing, "Design and Simulation of a SOA-based System of Systems for Automation in the Residential Sector," in *ICIT*, 2013.

[10] *UML Profile for Modeling and Analysis of Real-time and Embedded Systems (MARTE)*, Object Managment Group Std., Rev. 1.1, 2011.

[11] T. Haerder and A. Reuter, "Principles of Transaction-oriented Database Recovery," *ACM Comput. Surv.*, vol. 15, no. 4, pp. 287–317, 1983.

[12] *IEC 61131-3:2013. Programmable controllers - Part 3: Programming languages*, International Electrotechnical Commission Std., 2013.

[13] *IEC 61499-1:2012. Function blocks - Part 1: Architecture*, International Electrotechnical Commission Std., 2012.

[14] W. Zhang, W. A. Halang, and C. Dietrich, "Specification and Verification of Applications based on Function Blocks," in *Component-Based Software Development for Embedded Systems*. Springer, 2005.

[15] T. Kothmayr, A. Kemper, A. Scholz, and J. Heuer, "Schedule-based Service Choreographies for Real-Time Control Loops," in *ETFA*, 2015.

[16] M. Panunzio and T. Vardanega, "A component-based process with separation of concerns for the development of embedded real-time software systems," *Journal of Systems and Software*, vol. 96, 2014.

[17] *Specification of RTE*, AUTOSAR Std., Rev. 4.3.0, 2016.

[18] H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Springer Science & Business Media, 2011.