Towards self-reconfiguration of real-time communication within Cyber-Physical Systems

Jan Jatzkowski¹*, Bernd Kleinjohann²

¹Cooperative Computing & Communication Laboratory, University of Paderborn, Fuerstenallee 11, 33102 Paderborn, Germany

Abstract

Today, in domains like automation and robotics systems consist of various sensors and computation nodes. Due to the temporal dependency in quality of measured data, such Cyber-Physical Systems (CPS) commonly have real-time requirements on communication. In addition, these systems shall become more flexible and scalable, e.g., by adding new components to the CPS. This would be most suitable if a CPS could react to the presence of a new component and reconfigure itself to run afterwards with the new component integrated to the CPS. This capability is covered by the term Plug-and-Produce. In this paper, we propose a concept to enable Plug-and-Produce within a CPS whose network uses different communication media, e.g., Ethernet and CAN. In this context, we also introduce our three layered software architecture that supports the proposed concept for self-reconfiguration.

Keywords: Cyber-Physical Systems; distributed embedded systems; dynamic reconfiguration; plug-and-produce; real-time communication; real-time systems

1. Introduction

Nowadays, interaction between a variety of system components controlling physical entities becomes more and more important. In [1], Lee and Seshia define that Cyber-Physical Systems (CPS) are about the intersection of these
physical and cyber entities. Depending on the application domain, these systems require at least partial real-time communication and in case of dynamically adaptable systems, this even implies reliable reconfiguration of the communication, especially real-time communication. Although such reconfiguration is already manually practicable, this process is error-prone and strongly time consuming due to user interaction. To overcome these issues, self-reconfiguration of such systems is required. In automation domain, e.g., self-reconfiguration after modification of a production line promises not only to be less error-prone and time-consuming, but also enables a higher utilization of available system components and cost reduction.

To enable self-reconfiguration of real-time communication within CPS, we present a three layered software architecture based on the ISO/OSI reference model: Each node of the networked system has an application, middleware, and connectivity layer. While elements of the application layer represent particular functionality of a system component's application domain, the connectivity layer provides the interfaces to the communication channels provided by the hardware of a system component. The middleware layer connects the application and connectivity layer and is responsible for managing inter-node communication, i.e. communication between applications mapped on different nodes of a networked system.

In this paper, we focus on the middleware layer of our architecture, especially on automatic reconfiguration of real-time communication. Real-time communication is usually realized by a time-triggered approach, i.e. time slots are assigned to nodes that are only allowed to send information within these assigned slots. The proposed self-reconfiguration concept considers modifications like integration of new components to the overall system, i.e. Plug-and-Produce capabilities. Safe modifications are realized by means of pre-defined time slots that are reserved, e.g., for introduction of new system components into the CPS. When such a modification is registered by the CPS, a self-reconfiguration process is started within the middleware layer. During this reconfiguration, our approach considers the required communication from application layer and creates a new assignment of time slots to system components guaranteeing required real-time capabilities. While the self-reconfiguration process itself does not need to provide real-time capabilities, its result has to guarantee the real-time requirements of the new system configuration.

The remainder of this paper is structured as follows. In Section 2, fundamentals about real-time communication as well as Plug-and-Produce are presented that are used to compare the presented concept with related work that is described in Section 3. Our concept for self-reconfiguration is presented in Section 4. Finally, the paper is closed by a conclusion and outlook in Section 5.

2. Fundamentals

This section provides fundamentals about real-time communication and concepts of Plug-and-Produce. This will support the analysis of related work in Section 3 as well as classification of the proposed concept for self-reconfiguration presented in Section 4.

2.1. Real-Time Communication

Due to the interaction between physical and computational entities within a CPS, timing behavior of communication is important for correctness of results produced by a CPS. Not only the measurements of sensors and processing of these measured data but also the transmissions of these data have to be finished before a pre-defined deadline to achieve correct behavior of a CPS. Therefore, Kopetz states that “real time is an integrated part of the real world that cannot be abstracted away” [2]. In [3], Buttazzo distinguishes three levels of real-time depending on the consequences of missing a deadline: hard, firm, and soft real-time. In soft real-time systems, missing a deadline causes performance degradation, but results can still be useful for the system. In contrast, results of a firm real-time system are useless when missing a deadline and in hard real-time systems, missing a deadline may even cause catastrophic consequences on the controlled system. Consequently, different real-time systems can differ with respect to the consequences of missing a deadline although all of them are labeled real-time.

With respect to real-time communication, Kopetz defines a set of requirements including timeliness and flexibility [4]. Timeliness covers among others clock synchronization, i.e. a global time base is required. This way, observations like sensor measurements are guaranteed to be temporally relevant for the controlled system when processed by a computation node. Flexibility refers to the adaptation capabilities of real-time communication in case
of different supported system configurations that can change over time. It should be possible to add new sensors and/or computation nodes to a CPS without violating temporal guarantees of the original CPS. However, flexibility is limited by the bandwidth available by the given communication channels.

In the automation and robotic domain, communication is often based on Ethernet as well as Controller Area Network (CAN) as protocols to realize physical and data link layer of the standardized OSI 7-layer reference model. Ethernet and CAN both are capable of event-triggered communication. Since events can occur at an arbitrary point in time, i.e., randomly, no temporal guarantees can be provided in general. For instance, two sensors want to transmit their data at the same time via the same communication channel to the same computation node. In case of Ethernet, this would result in a collision that has to be handled by the protocol. In case of CAN, one of the sensor nodes’ messages would have a higher priority and thus postpone transmission of the other sensor data. This example shows that neither Ethernet nor CAN provide real-time capabilities in general.

However, there exist techniques to get real-time performance based on these protocols, e.g., switch technology improved Ethernet by eliminating collisions [5]. Examples of real-time Ethernet protocols built on top of standard Ethernet are PROFINET [6], EtherCAT [7], and Ethernet-Powerlink [8]. Based on CAN, e.g., the time-triggered CAN (TTCAN) protocol has been evolved to achieve real-time capabilities [9].

2.2. Plug-and-Produce

Plug-and-Produce is known from domains like automation and robotics. It is based on the Plug-and-Play technology that originally was developed for general purpose computers as used in office applications and is known, e.g., from the commonly used Universal Serial Bus (USB). Due to the domain-specific requirements of automation and robotics, the term Plug-and-Produce was introduced by the EU funded project SMErobot™ [10]. In [11], Naumann et al. focus on robot cells at shop floors and define Plug-and-Produce as the ability “to add devices to a robot cell and to use the functionality of these devices without the need of configuration”. Based on this definition, they define three Plug-and-Produce layers:

- **Application**: Offers automatically services to the user depending on the available functionality.
- **Configuration**: Configures default values, bandwidth requirements, etc.
- **Communication**: Deals with communication protocols and provides, e.g., discovery and addressing of devices.

These layers are hierarchically ordered: Plug-and-Produce on Application layer requires Plug-and-Produce on Configuration layer that in turn requires Plug-and-Produce on Communication layer.

To compare our concept with state-of-the-art approaches and highlight differences more precisely, we use an additional way to classify Plug-and-Produce respectively Plug-and-Play implementations. This classification depends on the system performance and is given by Zimmermann et al. [12]:

- **Cold**: The entire system is shut down, new components are connected and finally, the system is switched on again. Reconfigurations needed due to the modifications of the system are processed during start-up phase.
- **Hot**: Components are added to or removed from the system during runtime. However, running applications must not be disturbed by this process.
- **Coordinated**: Adding and removing of components is user or program controlled. This way, modification of a system is no longer enabled at an arbitrary point in time, but rather announced to the system. Thus, running applications cannot be disturbed by a randomly occurring reconfiguration process.

Based on these two complementary ways to classify Plug-and-Produce approaches, we consider related work in the next section.
3. Related Work

Different approaches for (re-)configuration of networks exist in literature, each focusing on different aspects of this problem. In [2], Kopetz and Bauer present the time-triggered architecture for real-time communication strongly focusing on fault tolerance, e.g., by means of different kinds of redundancy. Nevertheless, the authors do not address Plug-and-Play functionality, but rather refer to the fact that one of the considered protocols, namely TTP/A, provides such capabilities.

Reinhart et al. [13] address automatic reconfiguration of industrial Ethernet networks and present a five-step-model for a coordinated Plug-and-Produce within Ethernet-based networks. But in contrast to our concept that covers, e.g., CAN in addition to Ethernet communication, they only consider Ethernet as communication medium within a CPS.

In [14], Marau et al. present a middleware supporting reconfiguration of real-time networks. Their approach provides hard real-time guarantees and covers hot Plug-and-Play not only by means of adding new components to the system, but also removing of nodes. But in contrast to the concept presented in this paper they focus on Ethernet as communication medium and define their own communication protocol called FTT-SE. Thus, they do not capture standard communication protocols like PROFINET that are currently used in domains like automation and robotics.

4. Concept for Self-Reconfiguration

The goal of our proposed concept is to enable self-reconfiguration of a CPS where reconfiguration is activated by a Plug-and-Produce step. Here, a CPS may use different communication media for sub-systems, e.g., Ethernet and CAN. Therefore, our approach is also applicable to CPSs with gateways connecting different sub-systems.

An example of such a CPS is shown in Figure 1 where three robot cells are connected to each other via Ethernet. Furthermore, a camera for production monitoring and a supervising control entity are connected to this Ethernet network. Due to the production the first robot cell is additionally connected via CAN to sensors for measuring temperature and humidity. Moreover, the last robot cell is connected to an air pressure sensor via CAN. A user can interact with the CPS over an additional connection to the supervising control entity.

![Fig. 1. Example of a Cyber-Physical System. Sensors for measurement of temperature, humidity, and air pressure are connected to different robot cells via CAN. Robot cells are connected to each other as well as to a control node and a camera via Ethernet.](image-url)
4.1. Software Architecture Supporting Self-Reconfiguration

To provide a common base for communication within a CPS, we propose a three layered software architecture composed of an application layer, middleware, and connectivity layer. A schematic view to this architecture is shown in Figure 2. The goal of this architecture is to separate application functionalities from communication mechanisms in such a way that a functional unit only specifies its requirements for data transmission by means of Quality-of-Service (QoS) attributes. Within the connectivity layer a set of communication protocols is implemented, e.g. CAN and the Ethernet-based real-time capable protocol PROFINET. These communication protocols are also described by a set of attributes that provide information to the middleware about their specific QoS capabilities. This way, the middleware can compare the required QoS attributes requested by the application layer to the provided QoS capabilities given by the available communication protocols within the connectivity layer. Based on this comparison, the middleware can determine all appropriate communication protocols that provide the required QoS attributes and choose one of them, e.g., depending on the current network load.

While hard real-time requirements of running applications must be guaranteed, soft or non-real-time communication requests of applications might be acceptable to miss. Therefore, the middleware is enabled to give feedback to the application layer if such communication requests of running applications cannot be served, e.g., because limits of available bandwidth are reached. This is also useful when adding a new component to a CPS because the system must be enabled to cancel integration of a new component if its additional communication would disturb the correct timing and behavior of the original running CPS. In this case, the component that was aimed to be integrated into the CPS would not be started. Moreover, the middleware signals the application layer that integration of the new component failed.

Since applications only specify QoS attributes regarding data transmissions required to proceed correctly, the middleware is enabled to modify communication behavior as long as it keeps QoS requirements fulfilled. In principle, this even allows to switch communication of an application from one communication medium to another. Thus, load balancing for non-real-time communication can be realized, too.

In this paper we address self-reconfiguration of a CPS, e.g., required for Plug-and-Produce. Due to this, in the following we will focus on the middleware of the presented software architecture that manages communication within a CPS.

![Fig. 2. Three layered software architecture supporting self-reconfiguration.](image-url)
4.2. Abstraction of Communication Media within the Middleware

Separating applications from communication media is realized within the middleware of our software architecture by a uniform abstraction of all available communication media. Since we want to provide real-time capabilities, we define an abstract cycle-based time-triggered communication medium: Cycles with a fixed duration (cycle length) are specified where each cycle consists of three phases as depicted in Figure 3:

- **Phase 0** is used for synchronization and thus processes preparations for Phase 1.
- **Phase 1** is split into slots of predefined equal length to cover real-time communication.
- **Phase 2** can be used for event-triggered data transmissions, i.e. non-real-time communication.

Most real-time capable communication protocols are based on time-triggered approaches, i.e. these protocols have a similar structure to the proposed abstract communication medium shown in Figure 3. Consequently, on the one hand, parameters of the common abstract communication medium – e.g. cycle length, length of the different phases, and slot length within Phase 1 – have to depend on the real communication media within a CPS.

On the other hand, the global time base that is required for real-time communication within a CPS can be established based on the synchronization mechanisms already implemented by the underlying communication protocols. This supports establishing a global time base, but it will also cause additional jitter due to the need of synchronizing time bases of different protocols that already include jitter caused by the protocol specific synchronization.

4.3. Self-Reconfiguration to Enable Plug-and-Produce

As described in Section 2.2, different classes of Plug-and-Produce are known from literature. Here, we address a coordinated Plug-and-Produce, i.e. integration of a new component to the running CPS is processed at a point in time that is controlled by the running CPS itself. This way, we avoid to disturb the running CPS during its reconfiguration process.

The presence of a new component within a running CPS is noticed as follows: One slot within Phase 1 of the abstract communication medium is reserved for registration of a new component during runtime of the CPS. This way, we can guarantee that adding a new component can be noticed by the CPS within each cycle of the abstract communication medium. A newly added component has to signal its presence depending on the transmission paradigm of the underlying communication medium it has been added to.

Consider the case that a new computation entity for processing sensor data needs to be integrated into the CAN network of the first robot cell shown in Figure 1. Due to the publish-subscribe communication model used by CAN, a component newly connected to the CAN bus is enabled to read all sent messages. Consequently, the middleware of the added component is able to access synchronization messages transmitted during Phase 0 and synchronize itself with the CAN-based sub-system. Based on the established global time, the new component can send a self-description within the slot of Phase 1 that is reserved for registration of new components. The self-description of a component that is sent at first contains information about the provided functionality as well as required and provided data. Since data size of the self-description depends on provided functionality and data dependencies of a component, transmission of these data can require more than one slot. In such a case, transmission of self-description data is continued at the next cycle using the reserved slot for recognition of new components.

![Fig. 3. Cycle of the abstract communication medium.](image-url)
Consider the case of adding a new computation entity to the Ethernet-based network shown in Figure 1, e.g., to add image processing capabilities to the camera-based production monitoring. Here, we can procedure based on the five-step model for Plug-and-Produce introduced by Reinhart et al. [13]. After the new component has been physically connected to the Ethernet network, its presence has to be noticed by the running CPS. In [13], Reinhart et al. state that various methods are known to accomplish detecting new components within Ethernet-based networks, e.g., pings, broadcasts or IP scanners. Due to the way we proposed to register a new component within a CAN sub-system, we want that a CPS recognizes the presence of a new component within the reserved slot of Phase 1 also in case of an underlying Ethernet-based communication medium. Therefore, we propose to define a reserved address that must be initially used by a new component. This way, the CPS can ping this address to check presence of a new component. This ping has to be processed at the start of the pre-defined slot in Phase 1 that is reserved for registration within each cycle of the abstract time-triggered communication medium. Since the ping is processed at the beginning of the registration slot, the new component is able to transmit its self-description data after the ping has been received.

Now, that the CPS has recognized the presence of a new component independent of the underlying communication medium, reconfiguration of the communication can start. To keep the current real-time guarantees valid, one approach to integrate the newly added and recognized component into the running CPS is utilizing unused slots within Phase 1 for real-time communication required by the new component. But availability of any unused slots within Phase 1 does not imply that real-time communication requirements of the new component can be guaranteed. Therefore, the middleware has to check if the required real-time communication can be processed by using the available slots. In this case, the middleware will assign the corresponding slots for the new real-time data transmissions and initialize start-up phase of the newly added applications. Otherwise, the middleware gives an appropriate feedback to the application layer that a newly added component could not be started because no suitable reconfiguration was found. Those data transmissions that have no real-time requirements will be processed within Phase 2 of the abstract time-triggered communication medium. Since these data are not time-critical, the CPS will process correctly with respect to both, timing and behavior even if data transmissions are postponed to later cycles.

During the process of checking real-time guarantees, the original CPS keeps running. Hence, the proposed concept provides a coordinated Plug-and-Produce that does not disturb running applications of the original system.

5. Conclusion and Outlook

We presented a concept for self-reconfiguration of real-time communication within a CPS that can be composed of sub-systems using different communication media, e.g., Ethernet and CAN. The software architecture proposed in Section 4.1 supports coordinated Plug-and-Produce functionality based on a common abstract cycle-based time-triggered communication medium which ensures that adding a new component can be registered by the CPS within one cycle. We presented how registration of a new component can be processed during runtime when added to an Ethernet sub-system as well as a CAN sub-system.

In our future work, we want to prove the applicability of our concept for self-reconfiguration. For this purpose, we aim at building a demonstrator system including different communication media and implementing our proposed concept.

Acknowledgements

This work was partly funded by the German Ministry of Education and Research (BMBF) through the project “it's OWL – Intelligente Technische Systeme OstWestfalenLippe” (02PQ1021).

References