Specification and Design of an Industrial Manufacturing Middleware

Frederik Gosewehr, Jeffrey Wermann, Waldemar Borsych and Armando Walter Colombo
Department of Technology
University of Applied Sciences Emden/Leer, Germany
Email: {frederik.gosewehr, jeffrey.wermann, waldemar.borsych, armando.colombo}@hs-emden-leer.de

***Abstract*—The European innovation project PERFoRM (Production harmonizEd Reconfiguration of Flexible Robots and Machinery) is aiming for a harmonized integration of research results in the area of flexible and reconfigurable manufacturing systems. Based on the cyber-physical system (CPS) paradigm, existing technologies and concepts are researched and integrated in an architecture which is enabling the application of these new technologies in real industrial environments.**

To implement such a flexible cyber-physical system, one of the core requirements for each involved component is a harmonized communication, which enables the capability to collaborate with each other in an intelligent way. But especially when integrating multiple already existing production components into such a cyber-physical system, one of the major issues is to deal with the various communication protocols and data representations coming with each individual cyber-physical component.

To tackle this issue, the solution foreseen within PERFoRM's architecture is to use an integration platform, the PERFoRM Industrial Manufacturing Middleware, to enable all connected components to interact with each other through the Middleware and without having to implement new interfaces for each. This paper describes the basic requirements of such a Middleware and how it fits into the PERFoRM architecture and gives an overview about the internal design and functionality.

***Index Terms*—Middleware, Enterprise Service Bus, cyber-physical systems**

I. Introduction

The industrial manufacturing area is currently facing many important challenges, which forces companies to invest in new concepts and technologies for their manufacturing facilities. On one hand, every company is always driven by the desire to reduce their expenses related to the production. On the other hand, customers are becoming more demanding, requesting products with high customization while keeping low prices and high quality. Product life-cycles are getting shorter and in some areas a trend towards mass-customization is visible. From the perspective of the manufacturer, this leads to various challenges related to the production of the product. The characteristic mass-production lines are often to static to deal with this customization. This pushes manufacturers to invest in more flexible systems, which allow an easy reconfiguration on the shop floor while still maintaining an economically feasible throughput.

Another aspect related to this paradigm shift in production is the distribution of information in these systems. While traditional production systems were able to produce the same thing over and over again with little parametrization, these new production systems are now in need of extensive interfaces to allow for the change of programs and other process parameters multiple times per hour and even during runtime. Furthermore, the information about what needs to be produced needs to get from the customer to the machine, usually by being transmitted through an Enterprise Resource Planning (ERP) system and a Manufacturing Execution System (MES). A standardized way to communicate is essential to realize this new connectivity requirements. This enables a easy reconfiguration of the shop floor with little to no downtime when new production systems are integrated.

This need for a coherent horizontal and vertical integration and the increasing demand for flexibility and reconfigurability has made an impact on the research work within the last decade, since new scientific accomplishments in areas like artificial intelligence are providing interesting concepts which can be applied in the industrial world to tackle the upcoming challenges. Important solutions viable for industrial application include the cyber-physical systems paradigm and agent-based or holonic systems. Under the terminology "Industry 4.0" countries like Germany are funding research and innovation actions to facilitate the integration of these concepts into the industry, calling it the fourth industrial revolution. [1] One of the key aspects of this is the digitalization of production components.

Within the Factories-of-the-Futures program of Horizon 2020, the European Commission is investing in this area as well. One of the projects targeting these topics is PERFoRM, which specifically is aiming to harmonize existing research results from various projects into one architecture, which is applicable for industrial use and will be demonstrated within the project in various industrial use cases. A core component of this architecture is an Industrial Manufacturing Middleware, which is used as a integration platform to enable the communication between the various components foreseen within the architecture. Thus, the Middleware is the enabler for all of the aspects discussed above, making both an easy vertical and horizontal communication possible, allowing the parameterization of flexible production systems and integrating discovery mechanisms for easy reconfigurability.

This paper will first give an overview about basic underlying concepts for the Middleware in Section II, before specifying the functionality and capabilities envisioned for a PERFoRM compliant Middleware and how it acts within PERFoRM's

978-1-5386-0837-1/17/$31.00 ©2017 IEEE

architecture (Section III. In Section IV, these general functionalities will then be detailed and derived into a specific Middleware solution design. Section V will give a brief overview about first tests and results using the Middleware. A comprehensive summary about the presented results is given in Section VI, where also an outlook to future work is presented.

II. State-of-the-Art

This section is presenting the current state of the art in the area of flexible and reconfigurable production systems by giving an overview about Industrial Cyber Physical Systems and the concept of Service-oriented Architectures. It furthermore explains the function of a Middleware in general.

A. Industrial Cyber-Physical Systems

During the last years the number of cyber-physical systems (CPS) is increasing steadily. A CPS is a physical component with mechanic and/or electronic parts and a cyber component, which in general is a communication interface and software for certain intelligence [2]. When applying this principle to an industrial context, it is referred to as Industrial Cyber-Physical Systems (ICPS). ICPS allows machines to communicate with each other, react to events or to notify other production related areas (e.g. storage for supply or maintenance, in case of disturbances). In order to promote reconfigurability, to avoid hard wiring and to make plug and produce possible, components in ICPS production environment should expose information about themselves (e.g. sensor data, status, hardware and software information, etc.). This allows a participant to access a digital twin of such machine and calculate possibilities and limitations of that, to finally make a decision which component to use for a specific task. [3]

For collaboration of different ICPS components communication is necessary. Nowadays there is no standard protocol, path or principle of communication for indusstrial components and ICPS, which is making the realization of an ICPS more complicated. A possible solution to this problem is the implementation of a Middleware.

B. Service-Oriented Architecture

Service oriented Architecture (SoA) is a software concept in which processes or process chains are stored behind a simple service call. The basic idea is that the caller of a service (consumer) is not interested in the process itself but in the result. The interface of the service is provided to consumers in order to abstract the processes, which are running in the background. This service is a single, atomic task for the consumer. SoA is a very popular concept in IT and is getting increasingly interesting for industrial application. By following the SoA principle, information can be provided from the shop-floor up to manufacturing operation system levels directly, thus flattening the previously strict hierarchy.

With SoA, the consumer can also receive the information from a provider, compare these with the requirements and data of other providers of the same service, and decide which industrial component is suitable for the particular task. However, to use services the consumer has to use the same technology as the provider and needs to discover the available provider with its services. As there are multiple different SoA technologies available, which might need to operate together, it is necessary to create an integration platform, which supports all SoA based technologies.

SoA design and implementation was a major research task in previous international projects, namely SOCRADES [4], IMC-AESOP [5], ARUM [6], etc. Overall these concept have shown good results within industrial environments in terms of machine collaboration, production plant reconfigurability and performance.

C. Middleware

In Computer Science, the term “Middleware” describes a software layer or component, which allows different software applications to interact with each other. Typically, large systems consist of many different software applications, which – in most cases – do not provide specific interfaces to each other, but implement individual interfaces. A Middleware is used in these cases to make these interfaces fit together. This can be useful as an additional component on top of an operating system, which will link different applications in one computer together, but is also useful in networked systems, where the applications are running on separate hardware. The advantage for the developer of software is, that he no longer needs to implement specific interfaces to each software it needs to interact with, but only one interface, which the Middleware can handle (see Figure 1). [7]

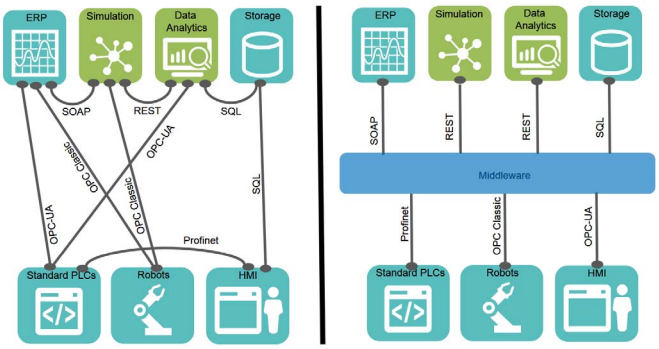

Fig. 1. Strong and loose coupled system interconnection with and without intermediate middleware

Middleware solutions are especially useful in areas, where a lot of different software applications need to work together and an easy integration of new components is important. The industrial world is currently undergoing a major shift, where it becomes increasingly important that each hardware component is expanded with an intelligent software component and a communication interface to form a cyber-physical component. An industrial environment can therefore offer a multitude of very different software systems, starting from low-level smart sensors and machines, to production planning and scheduling software, up to business planning systems. Therefore, Middle-

ware solutions are also becoming more and more necessary for industrial systems.

III. Middleware Specification

To apply these concepts into the PERFoRM environment, this section will discuss the PERFoRM architecture and how a Middleware is used within the architecture. The core functionalities of a PERFoRM compliant Middleware will then be specified.

A. PERFoRM Architecture

Figure 2 shows the PERFoRM system architecture as described in [8]. The architecture was built on the experiences and results from various previous research projects, such as SOCRADES, IMC-AESOP or ARUM. The architectural design is based on a service-oriented approach to match the requirements for flexibility and reconfigurability with distributed and heterogeneous hardware and software components, as envisioned within PERFoRM.

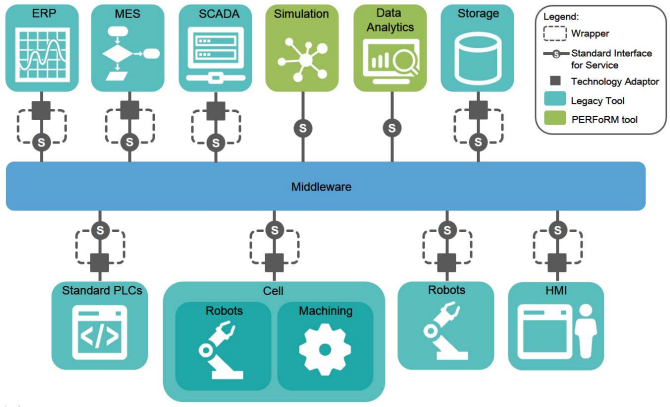

Fig. 2. PERFoRM Project architecture [8]

The architecture is representing different components of an industrial enterprise and how they are connected with each other. Within the architecture, the following architectural elements are foreseen:

- Legacy Tools
- Technology adapters
- Standard Interface
- PERFoRM-compliant Tools
- PERFoRM Middleware

The legacy tools are all hardware and software components, which are not developed directly within the PERFoRM project and therefore need to be integrated and adapted in some way to fit PERFoRM's architecture. Legacy tools can exist on the shop floor level, where typically PLCs, RCs and other machines need to be considered, but also on IT-level software, such as ERP or MES. For each of these tools, tool-specific adapters need to be developed to expose its data and interfaces in a PERFoRM compliant way, which is defined by the standard interfaces and the data model.

The standard interfaces are standardized within PERFoRM to have a defined way to describe how data can be accessed. They provide a set of services, which can be implemented by the adapters. Furthermore, a data model for the semantic description of data is specified. To find the right data model, an assessment of multiple popular already existing data models has been carried out and AutomationML was selected as the base for PERFoRM's data model (PML), which extends AutomationML with additional models required for PERFoRM [9].

These standard interfaces need to be implemented as a service, following the SoA paradigm. For the legacy tools, the technology adapters are attached with additional wrappers to a service interface. Tools, which are developed within the project directly are expected to implement the standard interfaces. As the service technology to use is not explicitly specified, a multitude of implementations is possible, e.g. REST, SOAP, MQTT or OPC-UA.

The role of the Middleware is to enable the communication between all of the aforementioned components, by ensuring a secure and reliable connection. Additionally it is acting as a broker/mediator between the communication partners, allowing components, which communicate using different protocols (e.g. REST and MQTT), to interact with each other.

B. PERFoRM Middleware features

After defining the general role of the Middleware in the context of PERFoRM's architecture, it is necessary to discuss the core functionality, which a PERFoRM-compliant Middleware is required to include. Figure 3 is showing the five main features of the Middleware with related detailed functionality.

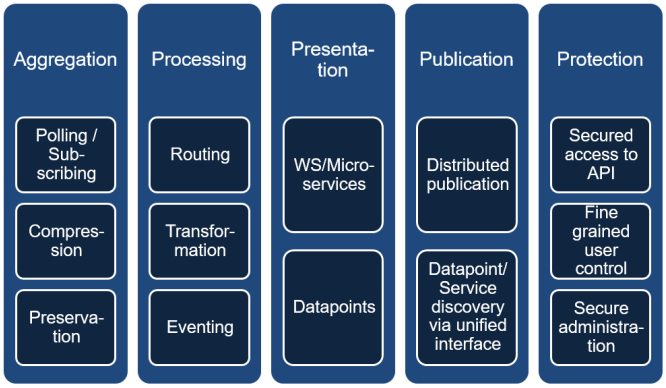

Fig. 3. Required middleware capabilities and features

1) Data Aggregation: Data Aggregation is the integral feature of the Middleware, meaning that it is able to send and receive data from various sources, such as hardware devices or software applications. Data acquisition methods such as polling data or subscribing to data changes need to be possible. The Middleware needs to be able to temporary buffer data during the transmission and to allow data compression by only storing important data.

2) Data Processing: After data has been received, the Middleware must be able to process said data before transmitting it to its destination. This includes basic routing functionalities,

where the Middleware has to be configured in a way which specifies the various destinations an incoming message needs to be send to. Furthermore, it must be possible to not only redirect the message, but also to transform the message. This is necessary, when protocols or data formats need to be translated.

3) Data Presentation: The Middleware has to be able to deal with various ways how data can be represented. As the PERFoRM Middleware is part of a SoA, the most important data presentation is in the form of Web services (e.g. REST or SOAP). Since the Middleware is used for industrial applications, it also requires direct access to data points, which are not represented as a web service, as it can be found in OPC Classic and OPC UA implementations.

4) Data Publication: A Middleware is targeting a loose coupled approach, where individual applications do not necessarily need to know details about each other. It additionally can provide services to publish and discover connected services. This means, that each component, which acts as a server and provides a service (either web service or a simple data point) can register said service within the Middleware. Other components are now able to discover this service by using discovery mechanisms. This way, applications can interact with each other dynamically and find each other during run-time without having to pre-configure to which exact communication partner it must connect to.

5) Data Protection: An important issue of all networked systems is the security. The Middleware must provide mechanisms to ensure a secure communication between all connected components. This includes the transmission security, which can be achieved by using various data encryption and other secure transmission methods. Furthermore, it is necessary to ensure controlled access to data, by being able to regulate which application can use a specific service or access a specific data point. Additionally, the Middleware itself must be secure, so that only restricted personnel can (remotely) re-configure the Middleware.

IV. Middleware Design

Considering previously mentioned aspects, a first evaluation of already existing Middleware solutions has been carried out and published in [10]. The most promising solutions are building the foundation of the specific Middleware design, which will be described in the following section.

A. Basic Middleware integration model

During the previous Middleware assessment [10] it became clear that there was no perfect fitting Middleware usable for any possible use case. Therefore instead of implementing all use cases on only one specific industrial Middleware system, a new approach was chosen with the goal to integrate arbitrary Middleware systems and concepts into a common external representation. This would allow for easy integration of any Middleware, as missing requirements are supplemented by the integration layer. As such integration layer is itself also only a service (of services), it could also run distributed throughout the network and hence would be able to integrate multiple different Middleware systems, e.g. one for the ERP- and another for the shop floor level, utilizing the same kind of external API and connection services for every connected system providing or requesting services. Figure 4 shows the whole model seen from an abstract perspective. As the whole system is a service oriented one, there is no defined upper and lower connection. All connected systems, supplying or requesting services, can both connect to the adapter services, which translate and transform the payload of a connected systems, e.g. a robot, into the PERFoRM compliant PML (PERFoRM Modeling Language) format or an incoming PML payload into the domain specific language of that system, e.g. robotic program language. Wiring these services together is the job of the Middleware and therefore the service data model, which is an extension of the PML, including web services and Middleware description into the shop floor level oriented PML data model. The whole approach aims for a data model driven configuration, be it automatic or manual. The following list defines the abstract methodology of the plug and produce concept when registering a new service provider:

1) The provider system announces its service- and data model to the Middleware.
2) Data model extension via manual inclusion at the right data model tree node by the operator.
3) Mapping or extension of the external API to wire the specific services to the external representation.
4) If needed, wiring of the adapters as intermediate processes for incoming data to extend/enrich, transform or translate the payload of connected service provider (although Figure 4 shows the adapter directly connected to a system, these adapters are in practice also services and can run anywhere throughout the network).
5) The services can be discovered through querying the service model API which returns a list of all Middleware/router routing endpoint addresses.

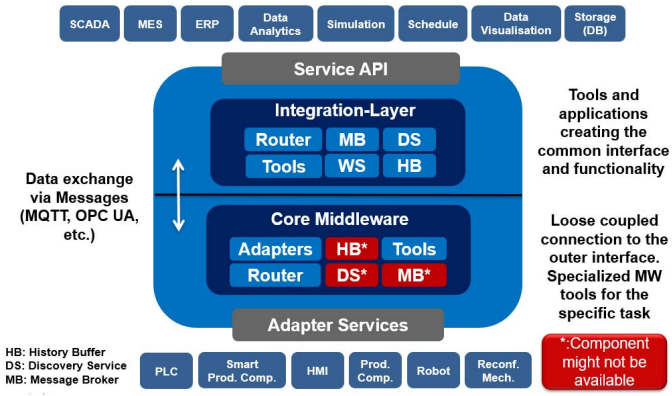

Fig. 4. Abstract Middleware integration model

B. Distributed approach utilizing Micro Services

The model has been created with a cloud distributed approach in mind, which allows to distribute all parts of the inte-

gration layer throughout the network, depicted as ”Integration Layer Service Cloud” in Figure 5. Such distribution of loosely coupled Micro-Services does not only enable for an easier maintain- and extendability but also includes currently used cloud service functions through the uses of technologies like Spring-Boot and Docker to encapsulate and isolate services from each other and Kubernetes and OpenStack to orchestrate these isolated containers within the infrastructure- and network topology. This allows for better scalability of processing power within the on-premises data center and therefore reduction of overall expansion costs.

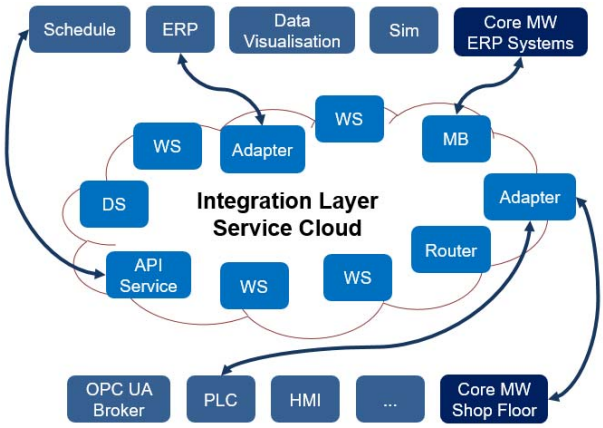

Fig. 5. Cloud based integration layer services following the Micro-Service approach

C. DaaS - Data-Model-as-a-Service

When looking at the data model, which is the heart of the PERFoRM architecture, as it defines both the connection of systems as well as the services provided via the extended service model, it would be bad design to strongly couple it with the integration layer/Middleware. Such design is vulnerable to failures as it creates a monolithic service provider where the failure of one service might results in the whole system failing. Therefore the PERFoRM data model uses a Micro-Service approach instead, decoupling the service API (the view) from the underlying persistence services (the business model). This decoupling guarantees that changes to the service API do not affect the persistence services and vice versa, which also reduces the services needed, as persistence calls, which follow the CRUD[1] principle, can be reused to generate compound services within the service API. The abstract model of the approach is depicted in Figure 6.

V. Test & Evaluation

To test the whole approach a test case, using a real UR10 robot, has been implemented. It is the tests overall goal to use a COLLADA CAD file to reprogram the robot while utilizing a service oriented architecture. This tests does showcase the loose coupled architecture, the resilient REST based Micro Services cloud as well as the data model driven configuration

[1]Create, Read, Update, Delete

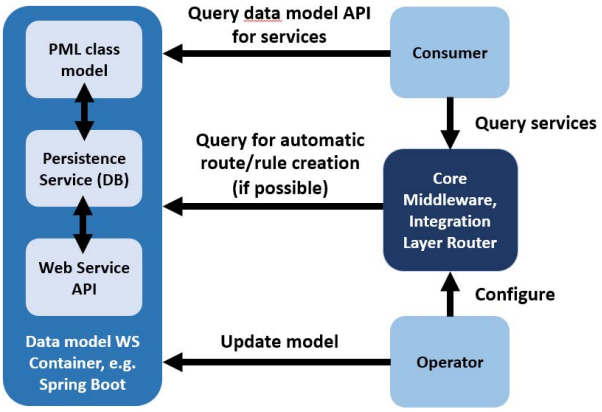

Fig. 6. Data model as object relational mapping service

and description of both services and payload data. Figure 7 depicts the different steps taken to query remote services which service calls have been abstracted, not showing the real REST calls, for readability purposes. Although the whole approach could be fully automated with no intermediate operator control, utilizing e.g. software agents instead, the test case includes the human control influence to also showcase the models capability to integrate the Human-in-the-Loop model. This is an important factor for critical systems where human supervision is mandatory for safety reasons.

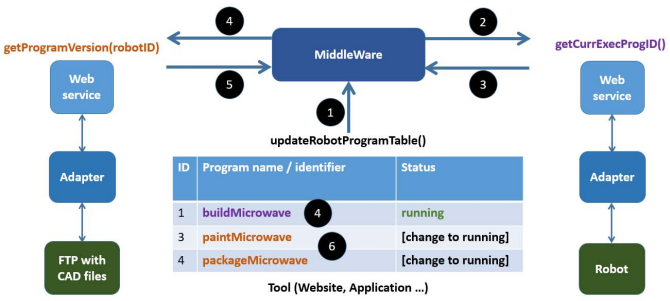

Fig. 7. Middlware test case, service oriented discovery of available and running programs

As depicted in the figure, the operator starts the process by choosing a new program to run. This internal process call does subsequently query the Middleware using its external service API. The figure does thereby only depict the immediate service calls. For simplicity, the service discovery to discover which service to use is not shown. This discovery is handled via querying the distributed Data-Model-as-a-a-Service service provider, which presents the PML data model as fully search- and queryable REST compliant database service, using object relational mapping (ORM) internally. This does allow the operators UI to filter for specific robots, e.g. with the same model version or connected to the same manufacturing cell.

After having chosen the robot, the UI automatically issues the *updateRobotProgramTable* service call, which is a compound service, composed of the services returning the current running program and all available programs (1). The first query

gets the identifier of the currently running program, which is represented by the external REST API as *robot/id/current*. This call is rerouted within the Middleware to the chosen robots service provider connected to the Middleware and mapped to the specific REST API call given (2). After returning the current running program identifier (3), the operator UI next queries the service to return the list of available programs. This call is rerouted to the FTP service provider (4), which forwards the query to the connected FTP adapter querying the FTP service defined in the provided service model. The response is a list of all program names usable with the given robot id (5).

When this process finishes, the operator is allowed to change the running program, indicated within the UI, to another one, as depicted in Figure 8. This figure does also show more of the actual implementation details, namely Apache ServiceMix, which acts as Middleware and Apache CXF, with which the web services, handling both SOAP and REST, have been defined. When executed the *updateRobotProgram* service queries the Middleware data model API with the robot and program id as parameter (1). The Middleware reroutes this query to the configured robot service provider (2) which itself queries the Middleware's robot program service *robot/id/program/id*, which in turn routes the query to the robot program service provider. This service subsequently returns the COLLADA CAD file encapsulated within an PML object, serialized as JSON or XML, back to the Middleware and from there to the robot service provider. The adapter responsible for the translation of the CAD file, given through the meta description within the PML CAD object, is then queried. This transforms the incoming CAD model into actual robotic control language, here for an Universal Robots UR10 industrial robot. If the program change was successful, the adapter returns a success message to the service, which in turn returns this message to the Middleware and from there back to the UI.

The whole setup is as loose coupled as possible to allow for an easy, data model driven reconfiguration even during operations. This makes it possible to maintain the whole approach using continuous integration and DevOps methodologies, enabling Developers as well as operators to react to changes quickly while maintaining software quality as well as ensuring the availability of services. Utilizing Micro-Services instead of monolithic ones ensures isolated test- and maintainability and also reduces the severity of service failures and unavailability.

VI. Conclusion & Outlook

This paper has described the key role of the PERFoRM Middleware within the PERFoRM architecture as an integration platform for the whole project, which is also applicable to other application areas. It illustrated the key features, which a Middleware solution must implement, to be compliant with PERFoRM's ideas and architectural design. The internal design of the Middleware, which is based on a distributed micro service approach and which is including existing Middleware solutions as a core is explained. The paper also

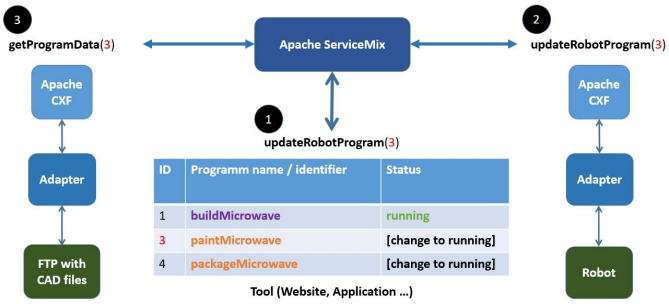

Fig. 8. Middlware test case, reconfiguration during runtime using service oriented approach

demonstrates the concept of the "Data-Model-as-a-Service" as a flexible approach to include PERFoRM's data model into the Middleware. Additionally, the Middleware design is evaluated with positive results in a simplified test case.

Upcoming steps for the Middleware implementation include extensive further testing of the concept itself and the performance of its various implementations. To evaluate the applicability to real industrial use cases, the Middleware will be instantiated in PERFoRM's test beds and use case scenarios.

Acknowledgment

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 680435.

References

[1] DIN Deutsches Institut für Normung e.V., "DIN SPEC 91345:2016-04: Referenzarchitekturmodell Industrie 4.0 (RAMI4.0)," Berlin, 2016.

[2] P. Leitão, A. W. Colombo, and S. Karnouskos, "Industrial automation based on cyber-physical systems technologies," *Comput. Ind.*, vol. 81, no. C, pp. 11–25, Sep. 2016. [Online]. Available: http://dx.doi.org/10.1016/j.compind.2015.08.004

[3] R. Rosen, G. von Wichert, G. Lo, and K. D. Bettenhausen, "About the importance of autonomy and digital twins for the future of manufacturing," *IFAC-PapersOnLine 48-3 (2015) 567–572*, 2015.

[4] A. W. Colombo and S. Karnouskos, "Towards the factory of the future: A service-oriented cross-layer infrastructure," in *ICT Shaping the World: A Scientific View*. European Telecommunications Standards Institute (ETSI), John Wiley and Sons, 2009, no. ISBN: 9780470741306, pp. 65–81.

[5] A. Colombo, *Industrial Cloud-Based Cyber-Physical Systems: The IMC-AESOP Approach*. s.l.: Springer International Publishing, 2014. [Online]. Available: http://lib.myilibrary.com/detail.asp?id=635192

[6] C. Marín, L. Mönch, P. Leitão, P. Vrba, D. Kazanskaia, V. Chepegin, L. Liu, and N. Mehandjiev, *A Conceptual Architecture Based on Intelligent Services for Manufacturing Support Systems, Man, and Cybernetics*, ser. SMC '13. United States: IEEE Computer Society, 2013, pp. 4749–4754.

[7] Q. Mahmoud, *Middleware for communications*. John Wiley & Sons, 2005.

[8] P. Leitão, J. Barbosa, A. Pereira, J. Barata, and A. W. Colombo, "Specification of the perform architecture for the seamless production system reconfiguration," in *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, Oct 2016, pp. 5729–5734.

[9] R. S. Peres, M. Parreira-Rocha, A. D. Rocha, J. Barbosa, P. Leitão, and J. Barata, "Selection of a data exchange format for industry 4.0 manufacturing systems," in *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, Oct 2016, pp. 5723–5728.

[10] F. Gosewehr, J. Wermann, and A. W. Colombo, "Assessment of industrial middleware technologies for the perform project," in *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, Oct 2016, pp. 5699–5704.