A Service Bus Concept for Modular and Adaptable PLC-Software

Virendra Ashiwal, Alois Zoitl, *Member, IEEE*
LIT CPS Lab, Johannes Kepler University Linz
Linz, Austria
{virendra.ashiwal, alois.zoitl}@jku.at

Matthias Konnerth
ENGEL Austria GmbH
Ludwig-Engel-Strasse 1
Schwertberg, Austria
Matthias.Konnerth@engel.at

Abstract—**How flexible and adaptive a production system can be, unquestionably depends on its software architecture and the level of modularity, flexibility, and adaptability of its software components. Programmable Logic Controllers (PLCs) are responsible for the functionalities of machines in the production system. Therefore PLC-software architecture and its software components play a vital role in a flexible, modular, and adaptive production system. In the current scenario, PLC software components use global variables to interact with each other to deliver certain machine functionality. Having global variables in PLC-software, increase hidden dependencies, reduces flexibility thus results in a less modular and less adaptable production system.**

In this paper, we are evaluating a service bus concept for modular and adaptable PLC-software that controls modular machines and production cells. As a result, a list of requirements for realizing such a concept is proposed.

Index Terms—**Factory Automation, control software, middleware, architecture, service Bus, PLC software, software components**

I. Introduction

Innovations from mainstream software engineering, with the increased computational power of embedded controllers, have been disrupting manufacturing industries since the beginning of the Twenty-First Century. This whole movement termed as a ”Fourth Industrial Revolution (Industry 4.0)” and challenged all stakeholders of the manufacturing industries to fulfill market demands such as customized products with better quality and shorter production times.

Programmable Logic Controllers (PLCs) are a crucial part of production systems for controlling and automating processes and mechatronics part of a machine. Software of PLC (PLC-software) composed of many software components which provide functionalities of a machine. Software components are often programmed with global variables as most of them depend on collaboration with other software components for fulfilling desired machine functionality. These global variables create a hidden data exchange structure which makes PLC-software structure more complex [1]. Because of such complex PLC-software structure, its software components are tightly coupled, less modular which results in reduced overall production system flexibility and adaptability.

In the late nineteenth century, Enterprise-level was also facing a similar issue with its available various different applications and the ”Enterprise Service Bus (ESB)” was used as a cure for it. Having such an ESB-like service bus concept for PLC- software components will let them communicate without using global variables, thus each software component will be modular and flexible for any further software development and support.

This paper gives an overview of the need for a ESB-like service bus concept for PLC-software components and propose its list of requirements for modular machines and production cells. In the next section, we will discuss the current status of PLC-software, which is a driving force for a new service bus type concept for PLC-software components. Background about the software integration concepts to improve modularity, flexibility and reusability are mentioned in Section III. In Section IV, software architectures based on middleware or service bus, available in Industrial automation are analyzed. A service bus concept and its list of requirements for modular and reusable PLC-software are proposed in Section V. Section VI discusses about the outcome of proposed concept. Finally, conclusion and future work is mentioned in Section VII.

II. Current status of PLC software architecture

A PLC-software is comprised of many software components responsible for the control and execute various functionalities of the modular machine and production cell. A software component is an atomic unit of the PLC-software and responsible for certain machine functionality. In this section, we are presenting the current scenario of PLC-software concerning modular machines and production cells.

1) Flexibility: PLC-software is comprised of software components and their interaction that provides overall machine functionality. The interaction is mainly done either via direct call or via indirect call(global variables) to read and write. Global variables create a hidden data exchange structure, which make the software structure even more complex [1]. Because of such a complex hidden structure, these software components are tightly coupled with limited modularity results overall reduced production plant’s flexibility.

2) Adaptability: Centralized PLCs architectures are economically cheap and often deployed to integrate the overall control logic of production cell. Such centralized architecture coordinates the execution of available machines and devices. Due to current market demands, these production cell are not suitable for short-time reconfiguration and demand a lot of manual efforts. It makes them less economically and time consuming [2]. Software can be changed more easily if each component is loosely coupled with high modularity.

3) High Maintenance Cost: Tightly coupled software components lead to strong dependencies and hard to maintain. It require a deep knowledge of software components, their functionality, and their interactions with others to avoid mistakes during maintenance. Global variables bring hidden dependencies. It make software comprehension and maintenance hard [3]. A change can spread among other software parts which could be hard to detect.

4) Inability to follow mainstream software trends: A PLC-software should follow mainstream software trends to improve software quality, connectivity, and user experience. Such adoption are costly and time consuming because traditional PLC-software is not prepare for that.

5) Interoperability: Modern production cell contains machines and robots with various PLC providers. For interaction with other available machines or robots in production cells, it is hard to maintain and implement a separate interface for each and every single available PLC in the market. Seamless exchange of information is a key requirement for future production system [2]. VDMA (Verband Deutscher Maschinen- und Anlagenbau) and OPC UA has a dedicated working group to bring in interoperability through standardized interface [4].

6) Portability: Although IEC 61131-3 is a common programming standard, it still requires significant effort to achieve portability to its full extent [5]. Mostly all production cells have controllers from multiple vendors, hence they have to maintain many dedicated software tools and responsible trained engineers. The software portability has potential to reduce these extra efforts and cost.

III. Available software integration concepts

Initially, software interaction was based on ”Point-to-Point Integration” between different systems and applications. It was based on a server-client principle, where server and client application had to know each other and also had to be present for communication. It is also known as ”spaghetti integration”. Maintainability of such a system was very low as for any new addition of system/application, all previously added applications needed to be updated. For communication mostly proprietary solution was used [6].

As a result, service-oriented architecture emerge to improve modularity and reuse of such applications. Later on for seamless integration between modular and distributed applications, SOA supported solutions such as middleware and ESB were emerged.

A. Service-oriented architecture (SOA)

Service-oriented architecture is a software concept, initially seen at traditional IT systems to improve system rigidness. Software development at production systems also got influenced by SOA. Applications based on SOA, are self-contained and loosely coupled. With this approach, the caller application does not need to know the implementation details of the connecting application, instead it just needs to know its service interface, which gives freedom for implementation. The main principles of SOA are:

a) Reusability [7]: SOA is a sustainable and economical approach for software design. Its services are autonomous and their operation is perceived as opaque by external applications. With opaqueness, SOA based application guarantees that external applications do not need to care its technical implementation. This implementation is encapsulated behind the self-defined interface which separates server and client providers. This way, SOA improve reusability of the software application.

b) Loose Coupling [8]: SOA services do not need to be instantiated before their use as it is needed for the implementation as per object-oriented. This creates an abstraction level for the client to use or reuse service provided by the server. It also allows the client, and server applications to be loosely coupled. Loosely coupled software applications are a desired and very important aspect for software architecture as it gives freedom for any further change in the software components without putting much efforts.

c) Discoverability [8]: Discoverability is a design principle for an individual service so that it becomes as discoverable as possible. it provide for example a service directory, service registry or a service search engine.

B. Middleware

Seamlessly information flows between different distributed applications where some of them are based on legacy information systems, require a complex integration process. This complex integration process leads to high maintenance cost, rigid integration and minimum flexibility for future add-on. An integration infrastructure such as middleware is necessary to increase connectivity and interoperability to overcome above said concerns [8]. It evolved from remote procedure call (RPC), via Object-Oriented/Component Middleware (OOCM) to message-oriented middleware (MoM) architecture. RPC suffered from problems such as inflexibility, tightly coupled, no object-oriented support and low scalability [9]. Object middleware, evolved from RPC by extending it by adding object-oriented concepts, but it is unable to resolve the scalability issue [10].

1) Message Oriented Middleware: MoM is a peer-to-peer model which allows distributed applications to interact via messages as shown in Fig. 1. Several features such as asynchronous and synchronous communication, message queues, data format transformation, and flexible integration of multiple systems are provided by MoM [9].

There are three major parts of MoM: Messages, Message queues and Message broker [11]. A message broker is a single point central message transfer managing unit, and thus a single point of failure [12].

2) Enterprise Service Bus: ESB is another middleware integration solution, where standardized interfaces are provided by each component, thus allowing service-oriented architecture approach to access data from each component. In this way,

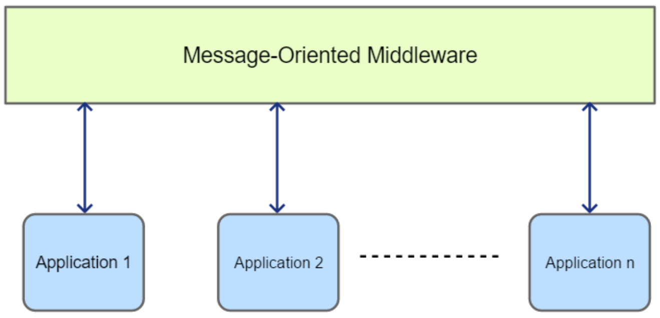

Fig. 1. Simplified architecture of MoM

typical hierarchies of distributed applications in the whole system are broken up and kept at the same level while allowing access to each other via their services [13].

As seen in Fig. 2, the main part of an ESB architecture is the Message-Oriented Middleware (MOM), a service container and management facility. MOM is the backbone of an ESB and is a distributed message server that allows reliable, secure, and manageable virtual channels for sending messages. The service container manages an application internally and, by using an adapter also provides access to an external application. MOM and service containers both are connected to a management facility so that all business services and virtual channels can be configured and monitored [14].

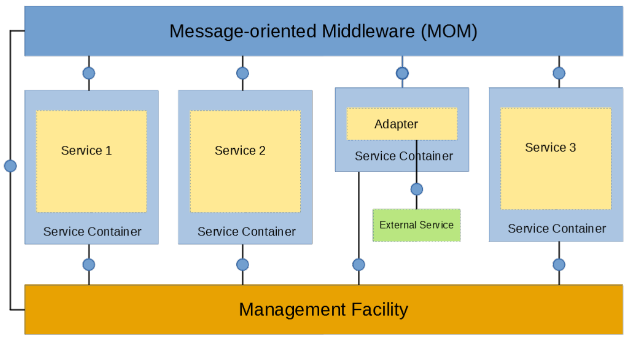

Fig. 2. ESB Architecture with its main parts [14]

Software components and their interactions from modular machines and production cells should be based on SOA and ESB type architecture concepts to achieve atomic and self-contained nature in order to increase reusability and modularity.

IV. Middleware and Service Bus Based Architectures in Industrial Automation

With the motivation from SOA, ESB, and middleware design patterns, several researchers worked on the software architectures for industrial automation. These available findings are presented in this section.

A. SOCRADES

According to [15], it is a SOA based middleware architecture. SOCRADES ensures that physical devices are available across different layers as service and offers device and service discovery, event management and storage, resource management and service composition. It also supports legacy devices via gateways or service mediators. According to [16], it does not fully support dynamic service composition and also security aspects are limited to authentication.

B. ARUM

Inspired by ESB, Adaptive Production Management (ARUM) is an agent-based middleware architecture. ARUM was mainly started for aircraft and shipbuilding industries to decrease risk of product immaturity and production disruptions. ARUM uses an intelligent ESB (iESB) with advanced modules, such as Ontology service, Data Transformation Service, Sniffer, Node Management and Life-Cycle Management. Several agent-based planning and scheduling tools along with legacy systems are integrated via a common infrastructure provided by the iESB. Several of these agent-based planning tools were developed under the scope of this project such as the Operational Scheduler (OS) and the Strategic planner (SP) and these are the core components of this project. The functionality of these tools is exposed as services and by ontology services, it is improving interoperability between such distributed and heterogeneous systems [17].

According to ANSI/ISA95, ARUM addresses interoperability at Level 3, where the agents' network is positioned, as well as a Level 4, supporting the integration with legacy systems [ibid] and does not address applications and device integration at control or even field-level of ANSI/ISA95.

C. Hufnagel & Vogel-Heuser

According to [18], this approach is a model-based data integration of distributed heterogeneous IT systems. It uses data mapping and adapters to transfer data from the various sources into a common information model with mapping rules. It enables transparency and data consistency across the enterprise because all data are accessible for all systems.

D. Line Information System Architecture (LISA)

LISA [19] is an event-driven architecture for loose coupling, a prototype-oriented information model and formalized transformation service. The main components of LISA are, a) a message bus, b) the LISA message format, c) the communication endpoints and d) the service endpoints.

LISA creates an event and transforms it into usable information in a loosely coupled manner. It uses a prototype-based approach which makes event creation, identification and filtering less rigid. LISA uses an ESB as a component that handles its message routing between distributed applications. An application can send event or information to Lisa's message bus (which is ESB) via the communication endpoints, here they are taking care of the translation of event and information as per LISA message format and publish LISA message to a respective channel of message bus.

For any hardware, variable or functionality change in application, only LISA's communication endpoints need to change . The addition of new application or device is really simple because of its communication adapters. It supports loose coupling, flexibility, and scalability. Original LISA uses JSON as its message format but some automotive industries did their own LISA implementations and used XML. It is also programming language independent which an advantage for the software development.

E. PERFoRM [20]

PERFoRM (Production harmonized Reconfiguration of Flexible Robots and Machinery) architecture is based on various research projects, such as SOCRADES, IMC-AESOP, and ARUM. It follows service-oriented architecture to match requirements for flexibility and reconfigurability with distributed and heterogeneous hardware and software components, such as PERFoRM Middleware, PERFoRM-compliant tools, standard interface, technology adapters, and legacy tools. These legacy tools are PLCs, robot cells (RCs), enterprise resource planning (ERP), manufacturing execution system (MES) and with the help of a tool-specific adapter, data are exposed in a PERFoRM compliant way.

A common data model for the semantic description of data is available, called "PERFoRM's data model (PML)" and implemented in AutomationML. This architecture is not bound to any specific service technology, so one can choose from available protocols (e.g. REST, SOAP, MQTT or OPC-UA).

Middleware is a communication enabler between all of the aforementioned components with a secure and reliable connection. It also acts as a mediator between different communication partners even if they use different protocols. The core features of PERFoRM middleware:

a) Data Aggregation: Middleware is able to send and receive data from various sources such as hardware (HW) or software (SW). Data acquisition methods such as polling or subscription-based are available. It also supports temporary buffer during transmission across various HW/SW components and also data compression.

b) Data Processing: Middleware supports routing functionalities, translation of message as per requirements at receiving end of middleware.

c) Data Representation: It support various data representations such as REST or SOAP (Web-based) or direct data points as OPC-UA node in case if those data points are not available as web services.

d) Data Publication: To be loosely coupled, the middleware also have a discovery mechanism, publish and discover for web service and also for a data point.

e) Data Protection: Middleware ensure secure communication between connected components with transmission security. It should also secure by itself.

PERFoRM architecture didn't mention its real-time communication capability, access to historical data in case of some data analysis components and there is also no concept about how inter-enterprise data exchange will take place [21].

F. IMPROVE

The innovative modeling approaches for production systems to raise validatable efficiency (IMPROVE) develop a decision-supporting system that could handle abnormality detection, diagnosis, and optimization in automated production systems. A virtual factory is created to concept, which serves as a base for model development and validation. This model can be adapted for current factory/process operation by having data from the real world for example when entire plant data need to be aggregated, integrated and provided [22]. IMPROVE also supports cross-organizational data exchange to improve machine/product quality. By cross-organizational data exchange, Machine manufacturers could develop models for lifetime prediction, improve production and increase overall Overall equipment effectiveness (OEE) by those shipped products [23].

IMPROVE is based on a MOM which uses a common information model, the so-called data management and integration broker, to achieve data integration from all level of the automation pyramid. A legacy system is also compatible via data adapters. A data Analyzer is part of this architecture, so data curation is available on the broker level, so that overhead can be minimized. The broker has also included access control and anonymization layer for any plant's outgoing data exchange. IMPROVE handles the quality of all incoming data from heterogeneous application (HW/SW) for the purpose of any further decision support, so suggestions for an improvement of product quality or minimizing the loss in the plant are available as outcome of IMPROVE data quality control but at the same time there are many operator level of expertise also exist so to match these improvement instructions to the level of operator expertise is still an open point.

IMPROVE lacks the capability of service detection and orchestration, and real-time communication [21].

G. BaSys 4.0

The changeability of production processes as one major goal of Industry 4.0 which was addressed by BaSys 4.0 [21]. It contains a Virtual Automation Bus (VAB) which enables cross-network peer-to-peer communication between shop floor and IT layer of ANSI/ISA95, Asset Administration Shells (AAS) which contain digital twins. With a virtual middleware bus, it aggregates all data and then uses in AAS to abstract the overall production process. The VAB consists of real-time communication for field and device layer and non-real-time for middleware to the IT layer, and are separated to ensure determinism.

"Service-based production" is one of the core concepts of BaSys 4.0. "Real-time production steps" of the PLC Controller are provided as re-usable service with the defined interface by a Process control component (PCC). PCCs control the invocation of these services. At the process level, real-time is not essential, allowing the service orchestrator to deploy these services to an Industrial PC or PLC. This way, rapid change of production system is possible by service orchestration and service parameters, although updating of PLC is only necessary when implementing new services [24].

H. Arrowhead Project [25]

Arrowhead is a SOA-based framework, which establishes the local cloud concept to meet automation system requirements, regarding (i) Real-time properties, (ii) Security and safety and (iii) Engineering of automation functionalities. A local cloud is defined as a closed group of industrial device/hardware within physical proximity. The local cloud always provides several basic core services enabling fundamental service-oriented properties such as service registration, service discovery, authentication and authorization plus the orchestration of a system of systems.

V. A SERVICE BUS CONCEPT FOR PLC-SOFTWARE

Software is an important part of any production system. According to [5], the software architecture which contains software components and their connections has a high influence on quality criteria, such as changeability, maintainability, or performance. All the in Section IV mentioned middleware architectures, try to reach all layers of ANSI/ISA 95 for seamless connectivity, including legacy component. IMPROVE tries to work with a decision supporting system and BaSys 4.0 with real-time data transportation.

All of them considered PLC-software as a single atomic software component which is part of the whole production plant's software architecture, but none of them considered PLC-software's components and their interactions as a part of the overall software architecture. Based on the current situation and problems, a new way for integrating PLC-software in an overall software architecture is necessary. The PLC-software components should be built as "atomic self-contained" and should offer their functionalities based on SOA principles. This is resembling how ESB has emerged to integrate heterogeneous distributed applications at the enterprise layer. Having an ESB-like service bus concept for PLC-software components will empower PLC-software to be modular, reusable, and self-contained.

A. Benefits the Service Bus Concept can Bring to PLC-Software

Here in this section, we will evaluate how a service bus concept for PLC-software will eliminate issues identified in Section II:

1) Flexibility: Service bus type software architecture doesn't support any point-to-point software integration, thus direct and indirect (global variables) interaction between PLC-software components won't occur. The absence of these kinds of interaction will help to overcome hidden dependencies between PLC-software components and improve flexibility and modularity.

2) Adaptability: To fulfill current market demands, PLC-software architecture must have the capability of short-time reconfiguration. The service bus concept based PLC-software is flexible, modular, and doesn't contain any hidden dependency within software components. Each software component can be independently updated to a new version if any new features are requested by the customer. Therefore it supports short-time reconfiguration. It is even possible to add a new machine or device in an existing production cell without disturbing the whole control logic of the production cell. For certain cases, such a change needs a safety certification so that the whole production cell gets back into production. It is an important aspect of adaptability but that's beyond the scope of this paper.

3) Reduced Maintenance cost: According to [26], when software becomes more complex, it tends to introduce more bugs hence the cost of maintaining such software increases. The service bus architecture enables its software components to be loosely coupled. Therefore each software component can be adapted independently towards new requirements or for an update. Software testing effort also get reduced with the absence of hidden dependencies. Hence it will ease to handle and maintain PLC-software throughout its life cycle.

4) Follow mainstream software trends: The short-time reconfiguration strongly supports new features in an exiting production cell based on available software trends. New features, inspired by software trends can be independently developed as software component without influencing the existing ones at PLC-software. If needed, the service bus brings advantage to provide a well-defined interface for the newly developed software component for further interaction with other software components. Such shorten development and testing time could motivate and help the PLC programmers to bring added values to the existing modular machines and production cells.

5) Improve Interoperability: ESB uses the service-oriented model to promote interoperability between its software components. Having such a concept will enforce to have a common information model at PLC-software architecture. VDMA and OPC UA foundation along with their industrial partners are working in this direction, and will definitely ease to implement such a common information model to bring interoperability in reality at PLC-software.

With all the above discussion, it is certain that having service bus concept for PLC-software will empower its self-contained software components and will improve flexibility, modularity, and adaptability of modular machines and production cells [4].

B. Proposed Concept

To understand the service bus like concept for PLC-software, we will take an injection molding machine (IMM) as seen in Fig. 3 as an example. The whole IMM is comprised of many mechatronic parts that are responsible for overall functionality but for simplicity, we will just consider the core elements of an IMM, such as heating, Ejector, and hopper unit. Here we will assume that a single PLC is used to control the whole IMM so the PLC-software contains all necessary

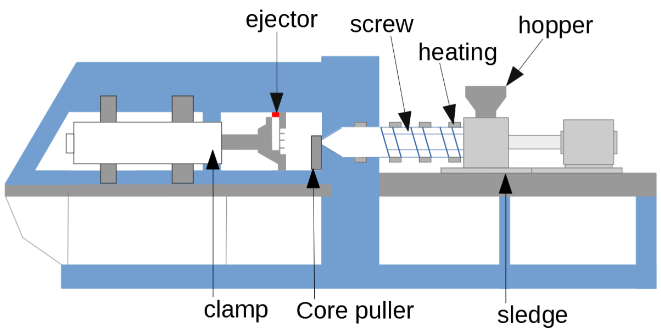

Fig. 3. Injection molding machine [27]

software components correspond to overall IMM functionality. Each machine functionality is programmed as a single software component. For example, Hopper is programmed in a software component named ”Hopper Unit” and likewise all others as seen in Fig. 4 with their initial version V1. Other than software correspond to mechatronics part of the IMM, PLC-software also has infrastructure software components such as alarm server, trend server, and many others. Each of the

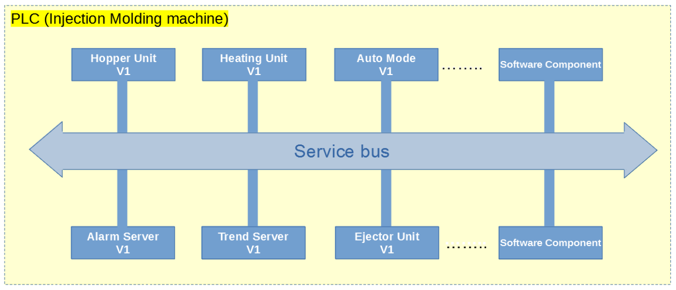

Fig. 4. Software architecture of IMM as an example

software components offers its services via the service bus. Initially for any new software component, it has to register itself at service bus. The service bus identify services offered by this newly added software component. An interface of services offered by all software component is uniform for all available services at the entire service bus so that a common understanding can be achieved.

For any future customer requirements which require respective software components to be updated, can be done independently. For example, a customer desired to update the ”Heating Unit” in order to get an improved heating-control algorithm. All heating related functionalities are already encapsulated in ”Heating Unit” as seen in Fig. 4. Therefore, a programmer just needs to update the ”Heating Unit” to a newer version V2 without disturbing any other software components as seen in Fig. 5. Efforts of software testing for such changes will also reduce compared to currently available PLC-software architectures for similar changes. Such a proposed architecture would make the software component an ”atomic self-contained’ unit in the PLC-software thus will improve its modularity and adaptability.

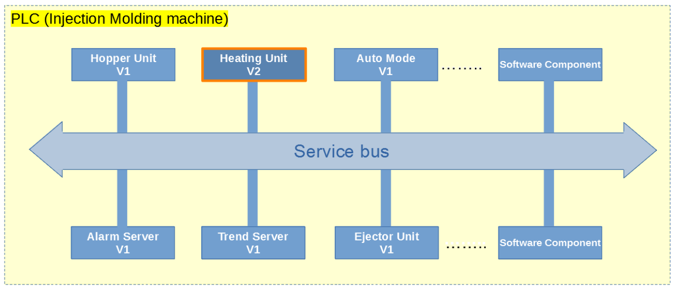

Fig. 5. Updated software architecture of IMM as an example

C. Requirements for Realizing the Proposed Service Bus Concept at PLC-Software Level

To realize such proposed service bus software architecture for PLC-software, it has to fulfill requirements that are crucial in the context of the industrial automation domain. These Requirements are derived from existing middleware and service bus based solution such as ESB, PERFoRM, and BaSys4.0. In this subsection, we will discuss them.

1) Handling of Software Components:

a) Registration of a Software Component: The proposed architecture should support registration of software components without being part of any system functionality. Once the software component is registered itself, its available services can combined with other available services in a composite manner. Such a composite service can be used to create a PLC-software functionality.

b) Registration of Services: A registered software components should be able to register its available services at a service registry of the service bus, so that it can be viewed and utilized by other software components.

c) Service Discovery: Service discovery is the automatic detection of devices and services available at the network. This functionality will reduce efforts for delivering certain machine functionality by service composition.

d) Decentralized Service Bus: Service bus has to be decentralised and go beyond a single PLC. This will create seamless interaction between decentralized PLC-software components as seen in Fig. 6.

2) Data Exchange Between Software Components:

a) Invocation Support: Initially software components registered its services with a service bus as aforementioned. ”Invocation support” is an ability of a service bus to send requests and receive responses from those registered services.

b) Message Prioritization: The proposed architecture should allow to transport messages with higher priorities within their defined time intervals.

c) Common Information Model: With increased involvement of data from various software components and with their different semantics, it is important to have a common understanding among them. Handling of them must be irrespective of their origin. Therefore, the proposed architecture must support a common information model for universal data transportation within the service bus.

d) Message Transformation: The proposed architecture must support message transformation. A message at its origin may support different transport protocol, message format, and message content which can be differ from the required one at its destination. Therefore, the proposed service bus architecture needs the capability to transform such a oriented message into the required destination format.

e) Communication: It is an important aspect for data exchange in overall production systems. For deterministic communication behaviour, it is better to separate real-time and non-real-time type communication. Real-Time: Data exchange at field level and control level for time critical applications. Non-Real-Time: To handle less time critical machine functionalities at PLC-software level so that they does not interfere with time critical applications.

3) Data Integration:

a) Support for Legacy Devices: Initial installation cost of the production system and machines are very high so they are desired to work for longer period of time which results in the requirement to provide support such a legacy devices for long period of time.

b) Adapter for Non-Native Communication: There are various industrial accepted protocols used in various machines and production systems. The proposed architecture should provide an adapter facility. With Adapters, those devices/applications which are not based on the native service bus communication protocol can be transformed to the main communication protocol of proposed service bus. The adapter concept bring seamlessly data flow without even thinking about available different protocols.

4) Security: With smart sensors and actuators the number of data they generate has increased tremendously. These data are further getting transported to PLC for various machine functionality. Therefore it demands that security aspects have to be included in the proposed architecture.

a) Authentication and authorization should be supported.
b) Encryption should be supported for data exchange via proposed service bus.
c) For technical support and remote data access, a secure connection should be supported by service bus.
d) Anonymous Access: In order to improve machine and product quality, a cross organizational data exchange is necessary. For example in case of a machine builder who would like to collect his sold machines performance data from its customers. In such cases, there has to be a process for a "secure anonymously access" so that identity of provider and some of its important data can be kept secret. This requirement is inspired by IMPROVE architecture as discussed in subsection IV-F.
e) The service bus itself needs to be secure.

5) Infrastructure Modules: PLC-software supports some infrastructure modules which are commonly used for all machines and production system such as alarms, trends, user management and data storage etc. Such infrastructure modules which are common for all PLC-software implementation, should be provided by a service bus by default.

6) Bus Management: Having a service bus also requires to manage its administration, data flow monitoring of all software components and available hardware and fault handling. These aforementioned management services must be part of service bus to insure its seamless operations.

VI. Discussion

Our evaluation confirms that a service bus concept for a PLC-software architecture would eliminate hidden dependencies and tightly coupled interaction among its software components. Therefore it improves modularity and adaptability of overall PLC-software. Such PLC-software could fulfill current market demands such as mass customization, better product quality with shortening production time.

All discussed middleware architectures in Section IV, are inspired by the service bus concept but none of them considered software components of PLC-software as part of it. There the PLC-software is considered as a monolithic software component.

There are also some limitations of the service bus concept. One major limitation is that the service bus, as central entity, introduces a single point of failure. It requires more experienced engineer for maintaining and there is a risk of introducing a regression. Having service bus architecture also introduce "application of service bus" itself along with PLC-software at PLC hardware, which could also affect its overall performance. According to [28] with modern emerging software technology like containers, distributed service bus, and with user-friendly user-interfaces aforementioned limitation can be reduced. But it is a matter of investigation and research in this direction.

Although these drawbacks needs further investigation we clearly see the benefits of service bus based architecture for future PLC software. Especially as it opens the PLC to allow an easy extension beyond a single PLC. Such a scenario is shown in Fig. 6. An Adapter concept needs to be created to realize such extension.

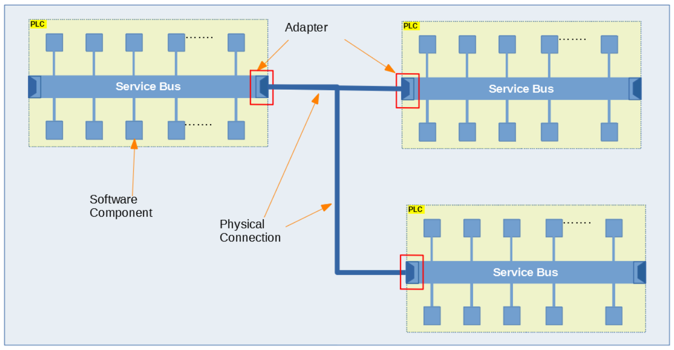

Fig. 6. Service bus concept extended for decentralized PLC

VII. Conclusion and Future Work

Flexible and adaptable production systems enable mass customization and shorten production time. Such production systems are designed for a lifetime of several years. PLC-software developments of such systems, make it difficult to manage because first, they need to provide complex system functionalities, and second, they need to keep high software quality to ensure maintainability and adaptability to be future-oriented.

The way software components are handled in PLC-software has significant impact to the aforementioned desired production systems. These software components are often interacting via either indirect call (global variables) or via direct calls. With the indirect call, it creates hidden dependency and with the direct call, it creates tight coupling and both are undesirable. Therefore the PLC-software becomes less flexible, less modular with reduced adaptability.

A service bus like software architecture provides key benefits to the software such as decoupling software components, transport protocol conversion, message Queues, message prioritization and transformation, and service orchestration. As discussed in Section IV, Several researchers adopted a service bus concept for industrial automation but none of them considered it for PLC-software. In the paper, such a service bus like concept is evaluated for PLC-software which shows great ability to overcome the current situation and the problems identified in Section II. After evaluation, a list of requirements such a proposed concept need to fulfill are discussed for modular machines and production cells as a target production system.

More research in this direction is needed to realize a working prototype. A technical specification is indeed the first step. With such a prototype we should do an experiment for its performance evaluation. Furthermore, new trends around service bus concepts and enterprise software integration should be evaluated and mapped to the PLC domain.

References

[1] J. Fuchs, S. Feldmann, C. Legat, and B. Vogel-Heuser, "Identification of design patterns for iec 61131-3 in machine and plant manufacturing," *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 6092 – 6097, 2014. 19th IFAC World Congress.

[2] N. Keddis, J. Burdalo, G. Kainz, and A. Zoitl, "Increasing the adaptability of manufacturing systems by using data-centric communication," in *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, pp. 1–8, 2014.

[3] Z. Yu and V. Rajlich, "Hidden dependencies in program comprehension and change propagation," in *Proceedings 9th International Workshop on Program Comprehension. IWPC 2001*, pp. 293–299, 2001.

[4] P. Zimmermann, E. Axmann, B. Brandenbourger, K. Dorofeev, A. Mankowski, and P. Zanini, "Skill-based engineering and control on field-device-level with opc ua," in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1101–1108, 2019.

[5] B. Vogel-Heuser, J. Fischer, S. Rösch, S. Feldmann, and S. Ulewicz, "Challenges for maintenance of plc-software and its related hardware for automated production systems: Selected industrial case studies," in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 362–371, 2015.

[6] A. Boyd, D. Noller, P. Peters, D. Salkeld, T. Thomasma, C. Gifford, S. Pike, and A. Smith, *SOA in Manufacturing Guidebook*. MESA: MESA International: IBM Corporation and Capgemini, may 2008.

[7] K. Channabasavaiah, E. Tuggle, and K. Holley, "Migrating to a service-oriented architecture, part 1." Available at https://www.ibm.com/developerworks/library/ws-migratesoa/ (2020/02/07), Dec 2003.

[8] J. Mínguez, *A service-oriented integration platform for flexible information provisioning in the real-time factory*. PhD thesis, University of Stuttgart, 2012. Available at http://dx.doi.org/10.18419/opus-2873 (2020/02/07).

[9] E. Curry, "Message-oriented middleware," in *Middleware for Communications* (Q. H. Mahmoud, ed.), pp. 1–28, Chichester, England: John Wiley and Sons, 2004.

[10] W. Emmerich, "Software engineering and middleware: A roadmap," in *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, (New York, NY, USA), p. 117–129, Association for Computing Machinery, 2000.

[11] G. Lilis and M. Kayal, "A secure and distributed message oriented middleware for smart building applications," *Automation in Construction*, vol. 86, pp. 163 – 175, 2018.

[12] K. Röppänen, "Requirements for an enterprise application integration tool," Master's thesis, 2013. Available athttps://trepo.tuni.fi/bitstream/handle/10024/84917/gradu06982.pdf?sequence=1(2020/02/07).

[13] 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*, pp. 5699–5704, 2016.

[14] M. Breest, "An introduction to the enterprise service bus," 2006.

[15] P. Spiess, S. Karnouskos, D. Guinard, D. Savio, O. Baecker, L. M. S. d. Souza, and V. Trifa, "Soa-based integration of the internet of things in enterprise services," in *2009 IEEE International Conference on Web Services*, pp. 968–975, 2009.

[16] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke, "Middleware for internet of things: A survey," *IEEE Internet of Things Journal*, vol. 3, no. 1, pp. 70–95, 2016.

[17] P. Leitão, J. Barbosa, M. C. Papadopoulou, and I. S. Venieris, "Standardization in cyber-physical systems: The arum case," in *2015 IEEE International Conference on Industrial Technology (ICIT)*, pp. 2988–2993, 2015.

[18] J. Hufnagel and B. Vogel-Heuser, "Data integration in manufacturing industry: Model-based integration of data distributed from erp to plc," in *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*, pp. 275–281, 2015.

[19] A. Theorin, K. Bengtsson, J. Provost, M. Lieder, C. Johnsson, T. Lundholm, and B. Lennartson, "An event-driven manufacturing information system architecture," *IFAC-PapersOnLine*, vol. 48, no. 3, pp. 547 – 554, 2015. 15th IFAC Symposium on Information Control Problems in Manufacturing.

[20] F. Gosewehr, J. Wermann, W. Borsych, and A. W. Colombo, "Specification and design of an industrial manufacturing middleware," in *2017 IEEE 15th International Conference on Industrial Informatics (INDIN)*, pp. 1160–1166, 2017.

[21] E. Trunzer, A. Calà, P. Leitão, M. Gepp, J. Kinghorst, A. Lüder, H. Schauerte, M. Reifferscheid, and B. Vogel-Heuser, "System architectures for industrie 4.0 applications," *Production Engineering*, vol. 13, pp. 247–257, Jun 2019.

[22] E. Trunzer, I. Kirchen, J. Folmer, G. Koltun, and B. Vogel-Heuser, "A flexible architecture for data mining from heterogeneous data sources in automated production systems," in *2017 IEEE International Conference on Industrial Technology (ICIT)*, pp. 1106–1111, 2017.

[23] E. Trunzer, S. Lötzerich, and B. Vogel-Heuser, *Concept and Implementation of a Software Architecture for Unifying Data Transfer in Automated Production Systems*, pp. 1–17. Berlin, Heidelberg: Springer Berlin Heidelberg, 2018.

[24] F. Schnicke, S. Sadikow, and T. Kuhn, "Basyx / benefits." Available at https://wiki.eclipse.org/BaSyx$_{/B}enefits$(2020/05/07).

[25] "Arrowhead framework technology architecture." Available at https://www.arrowhead.eu/arrowheadframework/this-is-it/architecture/ (2020/05/07).

[26] E. Ogheneovo, "On the relationship between software complexity and maintenance costs," *Journal of Computer and Communications*, vol. 02, pp. 1–16, 01 2014.

[27] A. Zoitl and H. Prähofer, "Guidelines and patterns for building hierarchical automation solutions in the iec 61499 modeling language," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 4, pp. 2387–2396, 2013.

[28] "Modern esbs: Pros cons." Available athttps://www.arcesb.com/blog/integration/20200114-modern-esb (2020/02/07), Jan 2020.