Native OPC UA Handling and IEC 61499 PLC Integration within the Arrowhead Framework

Jose Cabral, Kirill Dorofeev
fortiss GmbH
Research Institute of the Free State of Bavaria
Munich, Germany
{cabral,dorofeev}@fortiss.org

Pál Varga
Dept. of Telecommunications and Media Informatics
Budapest University of Technology and Economics
Budapest, Hungary
pvarga@tmit.bme.hu

***Abstract*—The Arrowhead framework enables interoperability and integrability for new and legacy systems by applying Service Oriented Architecture (SOA) principles at the Industrial Internet of Things (IIoT) domain. The core systems of Arrowhead provide service registration, discovery, security and other services for application systems within the System of Systems domain under their authority.**

While the application systems can communicate with each other using any protocol that fits them, the core Arrowhead services have been available as RESTful HTTP implementation only. This practically meant that those application systems without built-in HTTP protocol support, must have used translators to connect to the Arrowhead services.

The current paper describes a pragmatic enhancement for OPC Unified Architecture (OPC UA) endpoints, so they are now able to utilize Arrowhead core services natively. Moreover, the paper provides a solution for IEC 61499 Programmable Logic Controller (PLC) integration, and a practical use case description as proof of concept.

I. Introduction

Industrial automation is a turbulently changing area nowadays, due to the appearance of IIoT, the advancements of Cyber-Physical Systems (CPSs), and their extension with structures of business models by the Industry 4.0 (I4.0) movement. It is clear that industrial systems – especially production – requires improved flexibility in processes, which raises expectations on dynamic re-configuration, interoperability and integrability at both design and operation time [1]. Naturally, these expectations can only be fulfilled together with keeping the guarantees for high reliability, high availability, timeliness, quality, security and safety – as base requirements of industrial production. The related challenges are well-known for researchers and domain experts alike [2].

The Arrowhead framework [3] addresses interoperability and integrability issues with a SOA approach, by keeping in mind the traditional requirements of industrial automation, as well. It proposes both design and operational time concepts – together with their reference implementations. Its aim is to cover the gaps and allow the stakeholders (i.e., developers, vendors, integrators, operators) to move ahead with their Systems of Systems together, allowing legacy and new systems to be interoperable. The Arrowhead framework does not limit the protocols used among systems. However, the framework suggests to use protocol translators to access its core services for those systems not using natively the protocols of the current reference implementation. One of the contributions of this paper is to propose an extension for the Arrowhead core system reference implementation, allowing OPC UA endpoints to interact natively with the Arrowhead core.

The IEC 61499 is an international standard that addresses the topic of event-driven Function Blocks (FBs) for industrial process measurement and control systems [4]. It introduces an open architecture for distributed control systems, which is especially important for embedded systems with legacy elements. The control flow is modeled based on the ”event” approach; each event is emitted from an output of one FB and can be received at one or several inputs of other FBs. The IEC 61499 architecture aims at minimizing the efforts of developers in deploying automation software to various distributed architectures of hardware, with the support of the event-based communication mechanism itself [5].

The paper presents a solution for IEC 61499 PLC integration through Eclipse 4diac™[1] and a practical use case description as proof of concept.

Together, the OPC UA extension for the Arrowhead core systems on the one hand, and Arrowhead framework services support implemented for IEC 61499 FBs on the other hand provide all infrastructure needed in order to seamlessly integrate industrial PLCs into the framework and interact with them as with other Arrowhead service providers and/or consumers.

The rest of the paper is organized as follows. Section II provides basic insights into the Arrowhead framework, after which Section III summarizes the related work, narrowly focusing on the similar combination efforts related to IEC 61499, SOA, OPC UA and the Arrowhead framework. Section IV describes the OPC UA integration within the Arrowhead core systems, while Section V describes the related IEC 61499 PLC integration. A concrete, running example is demonstrated in Section VI, after which Section VII concludes the paper.

This work is funded by the Electronic Components and Systems for European Leadership Joint Undertaking (ECSEL JU) under grant no. 737459 through the project Productive4.0 (https://productive40.eu).

[1]https://www.eclipse.org/4diac/

II. Arrowhead Framework

The Arrowhead framework provides interoperability between services by defining a set of interfaces, rules and documentation system that allow systems and devices to provide and consume services following a SOA approach. The Arrowhead framework also defines types in an abstract way for Services, Cloud, Systems, System of Systems, and others concepts.

The framework establishes that there should be at least 3 main systems – also called mandatory core systems – in an Arrowhead local cloud:

- **Service Registry system**: allows to register, unregister and query for services;
- **Orchestration system**: allows to find provided services that were designated to devices to consume;
- **Authorization System**: manages all authorization needed between devices and services.

These systems, together with other core systems that are not mandatory (but also made available by the framework), and the actual application systems provide a set of services that are defined using the interfaces and types of the Arrowhead framework. Such an elementary, SOA-based Local Cloud – that consists of the three mandatory Core systems, some Applications systems, and their service-oriented approach – is shown in Fig. 1.

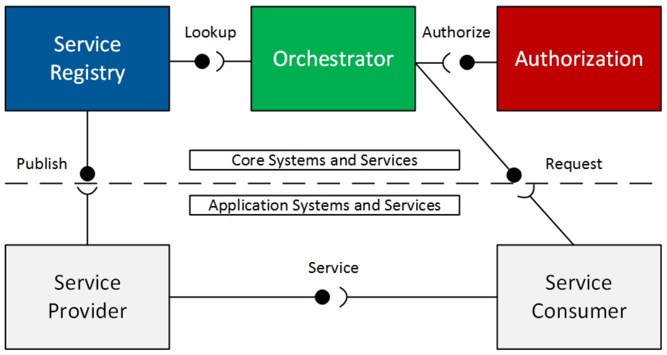

Fig. 1. Arrowhead mandatory core interactions [6]

The description of services is technology-independent to avoid any technological constraint [7]. The official implementation of the Arrowhead framework uses HTTP-JSON as interface for a service description [6], but the framework acts as a mediator between a producer and a consumer to support the SOA principles of *loose coupling*, *late binding* and *discovery* (or *lookup*, hence the triple "l"). This also means that the protocols and other specifications of a service between the actual *Application Systems* are not a part of the framework and needs to be handled by a consumer/producer.

The basic sequence for having a service being produced and consumed in the Arrowhead framework can be described as follows:

1) Configure the rules in the Authorization System to determine what services are allowed to be consumed by what system.
2) Configure the rules in the Orchestration System to determine where are the service providers for the consumers (IP address and port).
3) The service producer registers itself to the Service Registry.
4) The service consumer contacts the Orchestration to ask the endpoint of the service provider.
5) The Orchestration checks its stored rules and available producers, and checks with the Authorization System if the consumer is allowed to consume a service from a producer.
6) The Orchestration answers the consumer with the endpoint of the service producer.
7) The service consumer connects to the service producer and consumes the service.

Overall, Arrowhead framework gains the ability for both monitoring and control of the manufacturing processes by deploying and using a large scale of smart devices and profiting from the benefits of interconnected CPSs [8]. Arrowhead provides the required connectivity and integration at various levels of manufacturing systems. By developing the FBs for PLCs that execute the production at the shop floor, we further enlarge the number of supported device classes that can be directly connected to the Arrowhead framework and interact with the other connected systems in order to achieve the level of the interoperability required by I4.0.

III. Related Work

The integrated usage of OPC UA and IEC 61499 together with Arrowhead is a natural idea. Nevertheless, so far these have been elaborated as initial concepts (proven through demonstrations, of course), without enforcing improvements in the Arrowhead core system implementations.

The combination of IEC 61499 with SOA is comprehensively presented through various papers of Dai, Vyatkin, Christensen, Dubinin and their co-authors. The main findings are synthesized in [9], where the authors bridge SOA and IEC 61499 for flexibility and interoperability, and reference case studies for distributed control systems, as well.

Translation capabilities are available within the Arrowhead framework by the supporting core system called Translator. While translation among RESTful protocols are relatively easy to tackle, error handling is still not trivial [10]. Translation between publish-subscribe (e.g., MQTT) and request-response (e.g., RESTful HTTP or CoAP) is also challenging, although various implementations are already available. When it comes to OPC UA, the translation challenges and solutions, together with an Arrowhead Translator-based case study are described by Derhamy et. al. [11]. Due to their focus on translation, these papers do not address IEC 61499 integration.

Derhamy et. al. also provided a possible combination of Arrowhead services and IEC 61499 function blocks [5]. Their paper also includes a framework for integration of IEC 61499 within SOA-enabled IIoT. Nevertheless, their endpoints are using RESTful protocols such as CoAP and HTTP (but not

OPC UA) together with the Translator that supports access to the Arrowhead core services.

Lam and Haugen implemented OPC UA services in a SOA manner [12] through utilizing the Arrowhead Local Cloud concept. Their use-case even included inter-cloud communication, which means that many Arrowhead local clouds were involved in service interactions. Their approach, however, was that OPC UA endpoints were capable accessing the Service Registry, Orchestration and Authorization core systems through HTTP protocol capabilities at the endpoints.

IV. OPC UA FOR THE CORE SYSTEMS

OPC UA is a service-oriented machine-to-machine communication protocol mainly used in industrial automation and defined in the IEC 62541 standard [13]. A big advantage of using it is that OPC UA provides a semantic description of the data being transported. Its information model can be enriched by various companion specifications, allowing specific information models for different domains.

OPC UA is acknowledged as a uniform communication protocol that is ensuring interoperability and transparency demanded by I4.0 [14]. In order to provide a better support of the I4.0-complaint services and to enable service-based architectures at the industrial automation shop floor level, the Arrowhead framework should cater for OPC UA. As mentioned above, the Arrowhead core systems are designed to be protocol-independent but the first reference implementation was done using an HTTP REST API. Derhamy et. al. [15] as well as Rönnholm [16] proposed to use translators to convert OPC UA – as well as some other communication protocols – to the Arrowhead core systems. The similar approach is implemented by Karvonen[2].

In contrast to this solution, we propose to implement an OPC UA interface for the Arrowhead core systems. This enables pure OPC UA-capable Application Systems (service producers and consumers) to communicate directly with the Arrowhead core systems – without the need of translators. Fig. 2 demonstrates the difference among these approaches.

We have extended the core systems' implementation with the OPC UA functionalities, so that whenever the Arrowhead core systems are started, they automatically spawn an OPC UA server for each mandatory core system in parallel to the existing HTTP interfaces. These OPC UA servers provide endpoints, where the OPC UA devices can connect to and call the corresponding methods of the *ServiceRegistry*, *Orchestration*, *Authorization* and *EventHandler* core systems. All provided OPC UA methods have input parameters, which are JSON encoded objects that should be sent to the corresponding services as defined by Arrowhead core systems specification. As a result, each core system has its own OPC UA server started. In order to use the core systems over OPC UA, a device must connect to the corresponding core system server via a generic OPC UA client and trigger the required core system service. Each core system service is implemented according to the Arrowhead documentation and is represented in a form of an OPC UA method. The Uniform Resource Identifier (URI) of a service as defined in the documentation is the browse path in the OPC UA address space, relative to the */Root/Objects/* folder. The implementation of the OPC UA interface for the Arrowhead core systems is open-sourced[3]. By implementing OPC UA interface for the core systems, we showed that the Arrowhead framework can function in a protocol-independent way and the protocols, other than HTTP, can be supported natively.

V. PLC INTEGRATION

The IEC 61131 standard [17] defines programming languages for PLCs. These are the most common programming languages to develop the application software that implements the control of an automated process. Currently, in the domain of the software engineering for industrial systems, there is a shift from centralized to distributed control architectures to cope with the growing complexity of the software engineering process [18]. The IEC 61499 standard [19] was defined to model applications for distributed systems in a platform-independent way, a characteristic that can be harder to achieve using IEC 61131.

PLCs are the centerpiece of almost all industrial automation systems. Considering a PLC as a part of SOA, the PLC software components should be provided as encapsulated services with a well-defined generic interface [20]. By wrapping the existing PLC functionalities into services, a PLC can be accessed from outside as a service provider (i.e. for information exchange). On their side, the PLCs should support all required functionality to interact with SOA framework.

The integration of PLCs into the Arrowhead framework was done by implementing FBs according to the IEC 61499 standard in the open-source project Eclipse 4diac™ – an open-source implementation of IEC 61499. We have implemented the needed communication with the official implementation of the Arrowhead *ServiceRegistry*, *Orchestration*, *Authorization* and *EventHandler* core systems [3]. The library of FBs was made public in the source code repository of 4diac.

The work was done in three different parts:

1) 4diac FORTE, the runtime environment[4].
2) 4diac IDE, the integrated development environment[5].
3) 4diac examples, where unit tests are provided[6].

A. Implementation in 4diac FORTE

The Arrowhead data types that are required for the communication between the runtime and the framework, such as Cloud, Service and others were defined following the IEC 61499 standard and implemented in the 4diac FORTE. For example, the 4diac data type file, which defines the *ArrowheadCloud* data type in IEC 61499, is shown in Listing 1.

[2]https://github.com/arrowhead-f/client-opc-ua-rest

[3]https://github.com/arrowhead-f/core-java/

[4]https://git.eclipse.org/c/4diac/org.eclipse.4diac.forte.git

[5]https://git.eclipse.org/c/4diac/org.eclipse.4diac.ide.git

[6]https://git.eclipse.org/c/4diac/org.eclipse.4diac.examples.git

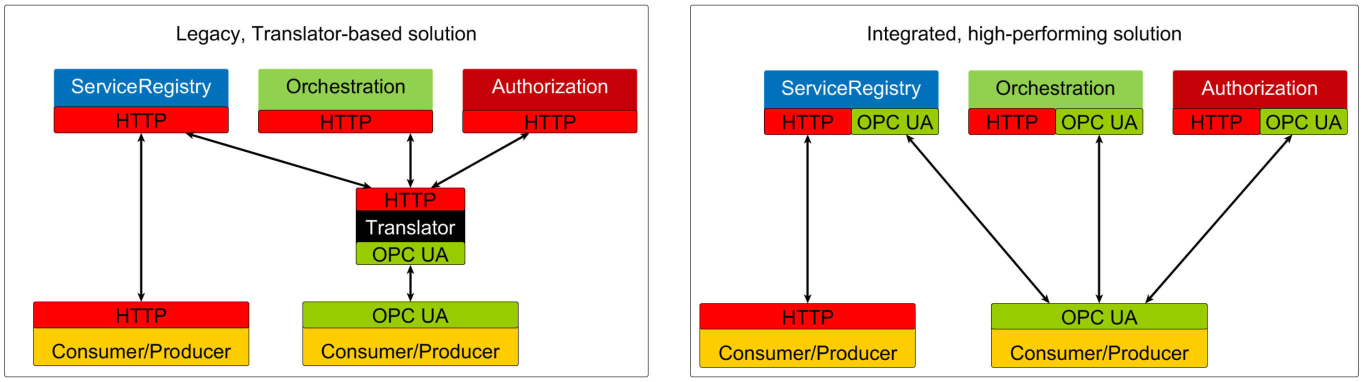

Fig. 2. Alternatives for OPC UA-capable systems to interconnect with Arrowhead core systems

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE DataType SYSTEM "http://www.holobloc.com/
    xml/DataType.dtd" >
<DataType Name="ArrowheadCloud" Comment="A cloud
    representation for the Arrowhead framework 4.0"
    >
<Identification Standard="61499-2" />
<VersionInfo Organization="fortiss GmbH" Version="
    1.0" Author="Jose Cabral" Date="2018-09-26" />
<ASN1Tag Class="APPLICATION" Number="1" />
<StructuredType >
<VarDeclaration Name="operator" Type="WSTRING"
    Comment="" />
<VarDeclaration Name="cloudName" Type="WSTRING"
    Comment="Name of the cloud" />
<VarDeclaration Name="address" Type="WSTRING"
    Comment="" />
<VarDeclaration Name="port" Type="DINT" Comment=""
    />
<VarDeclaration Name="gatekeeperServiceURI" Type="
    WSTRING" Comment="" />
<VarDeclaration Name="authenticationInfo" Type="
    WSTRING" Comment="" />
<VarDeclaration Name="secure" Type="BOOL" Comment=""
     />
</StructuredType>
</DataType>
```

Listing 1. Arrowhead Cloud type defined following the IEC 61499 standard

The implementation in 4diac FORTE also required its extension to support the communication protocols. HTTP and OPC UA communication layers already existed, but were improved and extended for better support. The transformation from IEC 61499 types to JSON, specific for the used protocols, was developed from scratch.

B. Implementation in the 4diac IDE and Unit Tests

The library of IEC 61499 FBs were defined in 4diac IDE. This was done in three logical levels. At the lowest level, the helper FBs allow user to create the arrowhead types using standard types from IEC 61499. Fig. 3 shows the FB that creates an instance of an *ArrowheadCloud* type, which can later be connected to the upper levels.

At the second level, the actual Arrowhead services are implemented. These FBs offer an adapter following the IEC 61499 standard, in order to decouple the abstract definition of services from the actual implementation (currently

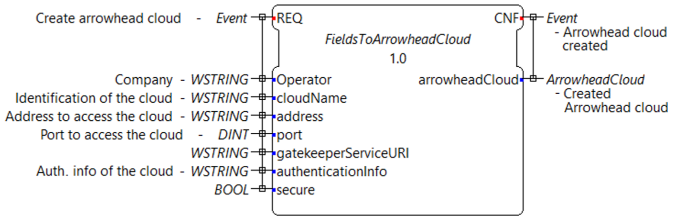
Create arrowhead cloud - Event
REQ
CNF
Event
- Arrowhead cloud created
FieldsToArrowheadCloud
1.0
Company - WSTRING
Operator
Identification of the cloud - WSTRING
cloudName
Address to access the cloud - WSTRING
address
Port to access the cloud - DINT
port
WSTRING
gatekeeperServiceURI
Auth. info of the cloud - WSTRING
authenticationInfo
BOOL
secure
arrowheadCloud
ArrowheadCloud
- Created Arrowhead cloud

Fig. 3. A helper FB for an ArrowheadCloud data type

available for HTTP Rest and OPC UA). As an example, Fig. 4 shows the FB to (un)register a service in the Arrowhead *ServiceRegistry* core system. The data inputs are a *ServiceRegistryEntry* (protocol-independent) and a *String*, specifying an endpoint of the *ServiceRegistry* core system. The *registerService* adapter on the below right side, offers a plug to the actual implementation of the communication, passing all needed data.

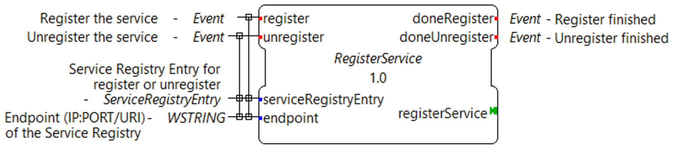
Register the service - Event
register
Unregister the service - Event
unregister
doneRegister
doneUnregister
Event - Register finished
Event - Unregister finished
RegisterService
1.0
Service Registry Entry for register or unregister - ServiceRegistryEntry
serviceRegistryEntry
Endpoint (IP:PORT/URI) of the Service Registry - WSTRING
endpoint
registerService

Fig. 4. Abstract definition of the Register Service offered by the ServiceRegistry core system

The corresponding adapter sockets are implemented for different protocols, respectively. Fig. 5 shows the FB that implements the actual HTTP communication to the *ServiceRegistry*. Its only input is the socket adapter, the counter part of the plug adapter from Fig. 4. By connecting them, the information is passed to the FB that handles the communication between a PLC and Arrowhead framework.

This decoupled architecture allows to simply extend the IEC 61499 implementation to support another type of communication protocol. For example, for the *ServiceRegistry* core system, which offers an OPC UA interface, only the FB in

RegisterServiceHTTP
1.0
registerService

Fig. 5. Register Service implementation in HTTP Rest

Fig. 5 needed to be re-implemented with the specifics of OPC UA.

At the top most level, the IEC 61499 sub-applications were implemented in order to facilitate the user using the library. For example, for registering a service, instead of building from the lowest level, which requires many FBs and connections between them, an encapsulated sub-application is provided that offers all needed parameters. Fig. 6 shows the sub-application that can be easily parameterized to register and/or unregister a service.

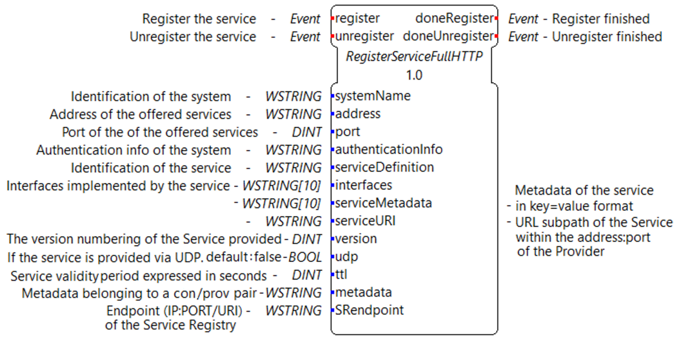
Register the service - Event
Unregister the service - Event
register
unregister
doneRegister
doneUnregister
Event - Register finished
Event - Unregister finished
RegisterServiceFullHTTP
1.0
Identification of the system - WSTRING
Address of the offered services - WSTRING
Port of the of the offered services - DINT
Authentication info of the system - WSTRING
Identification of the service - WSTRING
Interfaces implemented by the service - WSTRING[10]
- WSTRING[10]
- WSTRING
The version numbering of the Service provided - DINT
If the service is provided via UDP. default:false - BOOL
Service validity period expressed in seconds - DINT
Metadata belonging to a con/prov pair - WSTRING
Endpoint (IP:PORT/URI) - WSTRING
of the Service Registry
systemName
address
port
authenticationInfo
serviceDefinition
interfaces
serviceMetadata
serviceURI
version
udp
ttl
metadata
SRendpoint
Metadata of the service
- in key=value format
- URL subpath of the Service
within the address:port
of the Provider

Fig. 6. Sub-application for registering a service using HTTP

In order to test and assure that the library performs as expected, a set of tests were implemented that enable the automatic testing of the code. The tests are generated from applications implemented in 4diac IDE and help to identify bugs both in the 4diac and Arrowhead implementations.

The developed FBs work with both – classical HTTP and implemented by us and described in Section IV OPC UA – interfaces of the Arrowhead core systems. This is useful, as not every PLC supports HTTP, while OPC UA is de facto standard for the data connectivity in the industrial automation domain [21].

VI. Running Example

To test the integration of PLCs within the Arrowhead framework, we developed an example following the architecture shown in Fig. 7.

The demonstrator consists out of:

- Two BeagleBones[7] that simulate traffic lights systems offering a *TrafficLight* service;
- A Supervisory Control and Data Acquisition (SCADA) system running on a mobile phone that consumes the *TrafficLight* service and shows the current state of one of the traffic lights on the screen;
- A monitoring application that checks the connection status of both traffic lights systems.

[7]https://beagleboard.org/bone

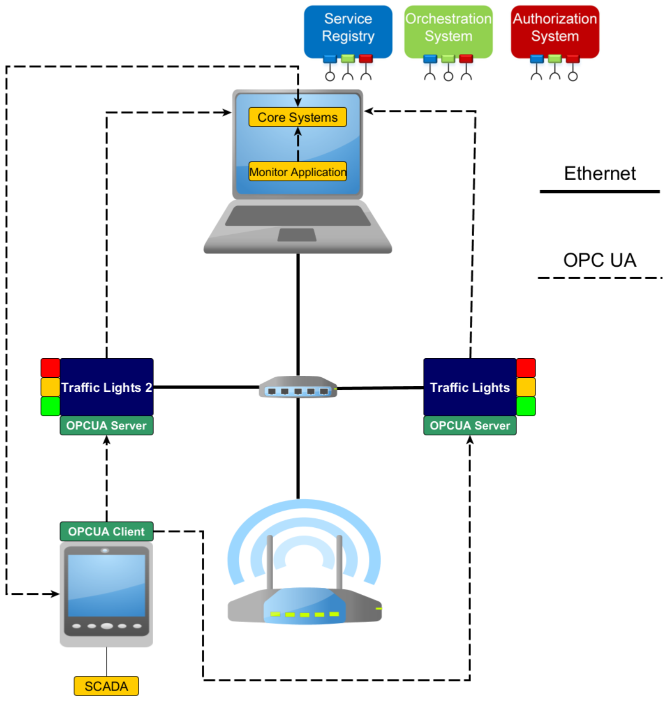

Fig. 7. Architecture of the example

The core systems offer OPC UA interfaces, and every connection to the core systems from the devices is done using OPC UA. The *TrafficLight* service running on a BeagleBone is offered by an OPC UA server.

The sequence of messages in the example is shown in Fig. 8:

1) Each traffic light system registers its *TrafficLight* service to the Arrowhead *ServiceRegistry* core system, after the system is started.
2) The SCADA system subscribes to a *newDevice* event of the Arrowhead *EventHandler* core system. After one of the *TrafficLight* services becomes available, the SCADA system consumes it and shows the current status of the lights on the screen of the mobile phone.
3) When the monitoring application detects that the currently used traffic light system is disconnected, it starts looking for another one, offering the same *TrafficLight* service.
4) When found, the monitoring application changes the orchestration rules to the new device.
5) Then, the monitoring application sends a *newDevice* event to the EventHandler.
6) The SCADA system receives the *new Device* event.
7) The SCADA system requests for a new orchestration rule.

8) The Arrowhead *Orchestration* core system sends the rule, which now contains the endpoint of the new device.
9) The SCADA system connects to the new device and starts consuming the *TrafficLight* service from it.

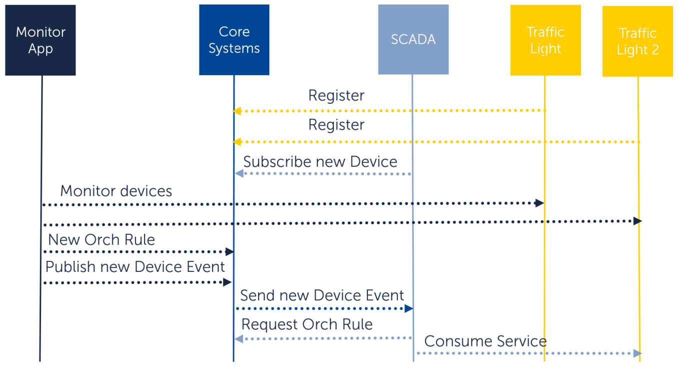

Fig. 8. Sequence of messages

The example shows the loose coupling between the service providers (traffic light systems) and consumers (SCADA). The SCADA system does not have any information about the traffic light systems, only about the interface of the *TrafficLight* service. The Arrowhead core systems communicate with the other systems only using OPC UA, except when the EventHandler sends a message (HTTP in this case), because this mechanism is not defined in the Arrowhead framework. The only intention of the example is to show the successful integration of PLCs into the Arrowhead framework, and not to offer a meaningful use case. The example is compatible with version 4.1.2 of the official implementation of the Arrowhead framework and not yet with the latest version 4.1.3 as the abstract interfaces tailored to HTTP were changed with the latest updates. The latter created barriers to use of other protocols such as OPC UA.

VII. Conclusions

The paper has described a solution for OPC UA-capable systems to natively interact with the Arrowhead Framework core, hence their service registration, authorization, discovery and orchestration do not require the use of translators. The system architecture was designed and developed with Eclipse 4diac™, an open source infrastructure for distributed industrial process measurement and control systems based on the IEC 61499 standard.

The paper provided the proof of concept via an IEC 61499 PLC integration, describing the FBs created for interaction with the Arrowhead framework. The System of Systems created in the running example included the PLCs directly communicating with the Arrowhead framework, and demonstrated loose coupling, late binding, and lookup (discovery) – three elementary concepts of the SOA approach.

References

[1] J. Delsing, “Local Cloud Internet of Things Automation: Technology and Business Model Features of Distributed Internet of Things Automation Solutions,” *IEEE Industrial Electronics Magazine*, vol. 11, no. 4, pp. 8–21, Dec 2017.
[2] P. Leitão, A. W. Colombo, and S. Karnouskos, “Industrial automation based on cyber-physical systems technologies: Prototype implementations and challenges,” *Computers in Industry*, vol. 81, pp. 11 – 25, 2016, emerging ICT concepts for smart, safe and sustainable industrial systems. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0166361515300348
[3] P. Varga, F. Blomstedt, L. Ferreira, J. Eliasson, M. Johansson, J. Delsing, and I. Martínez de Soria, “Making system of systems interoperable – The core components of the arrowhead framework,” *Journal of Network and Computer Applications*, vol. 81, 08 2016.
[4] V. Vyatkin, Ed., *Distributed Control Applications: Guidelines, Design Patterns, and Application Examples with the IEC 61499*. Instrumentation Society of America; 3rd ed., 2014.
[5] H. Derhamy, D. Drozdov, S. Patil, J. van Deventer, J. Eliasson, and V. Vyatkin, “Orchestration of Arrowhead services using IEC 61499: Distributed automation case study,” in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sep. 2016, pp. 1–5.
[6] C. Hegedus, D. Kozma, G. Soos, and P. Varga, “Enhancements of the Arrowhead Framework to Refine Inter-cloud Service Interactions,” 10 2016.
[7] F. Blomstedt, L. L. Ferreira, M. Klisics, C. Chrysoulas, I. M. de Soria, B. Morin, A. Zabasta, J. Eliasson, M. Johansson, and P. Varga, “The arrowhead approach for SOA application development and documentation,” in *IECON 2014 - 40th Annual Conference of the IEEE Industrial Electronics Society*, Oct 2014, pp. 2631–2637.
[8] J. Delsing, *IoT Automation: Arrowhead Framework*, 2017.
[9] W. Dai, V. Vyatkin, J. H. Christensen, and V. N. Dubinin, “Bridging service-oriented architecture and iec 61499 for flexibility and interoperability,” *IEEE Transactions on Industrial Informatics*, vol. 11, no. 3, pp. 771–781, June 2015.
[10] H. Derhamy, J. Eliasson, J. Delsing, P. P. Pereira, and P. Varga, “Translation error handling for multi-protocol SOA systems,” in *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*, Sep. 2015, pp. 1–8.
[11] H. Derhamy, J. Rönnholm, J. Delsing, J. Eliasson, and J. van Deventer, “Protocol interoperability of OPC UA in service oriented architectures,” in *2017 IEEE 15th International Conference on Industrial Informatics (INDIN)*, July 2017, pp. 44–50.
[12] A. N. Lam and O. Haugen, “Implementing OPC-UA services for Industrial Cyber-Physical Systems in Service-Oriented Architecture,” in *IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society*, vol. 1, Oct 2019, pp. 5486–5492.
[13] International Electrotechnical Commission, “IEC TR 62541-1:2016 - OPC unified architecture - Part 1: Overview and concepts ,” 2016.
[14] Plattform Industrie 4.0 – 2018 Progress Report, “Applying Industrie 4.0. Forward Thinking. Practical. Connected.” Apr 2018.
[15] H. Derhamy, J. Eliasson, and J. Delsing, “IoT Interoperability—On-Demand and Low Latency Transparent Multiprotocol Translator,” *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1754–1763, Oct 2017.
[16] J. Rönnholm, “Integration of OPC Unified Architecture with IIoT Communication Protocols in an Arrowhead Translator,” 2018. [Online]. Available: https://ltu.diva-portal.org/smash/get/diva2:1238235/FULLTEXT01.pdf
[17] International Electrotechnical Commission, “IEC IEC 61131-3:2013 Programmable controllers - Part 3: Programming languages,” 2013.
[18] A. Zoitl and T. Strasser, Ed., *Distributed Control Applications: Guidelines, Design Patterns, and Application Examples with the IEC 61499*. CRC Press, 2016.
[19] International Electrotechnical Commission, “IEC 61499-1:2012 Function blocks - Part 1: Architecture,” 2012.
[20] L. Ollinger, A. Abdo, D. Zühlke, and H. Heutger, “SOA-PLC – Dynamic Generation and Deployment of Web Services on a Programmable Logic Controller,” *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 2622 – 2627, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1474667016420057
[21] C. Resnick and D. Clayton, “OPC Technology Well-positioned for Further Growth in Tomorrow’s Connected World,” 2018. [Online]. Available: https://opcfoundation.org/wp-content/uploads/2018/02/ARC-Report-OPC-Installed-Base-Insights.pdf