Arrowhead Datamanager integration with Eclipse 4DIAC environment

Marco Romanato∗, Dmitrii Drozdov∗, Sandeep Patil∗, Jerker Delsing ∗, and Valeriy Vyatkin†
∗Department of Computer Science, Computer and Space Engineering, Lulea Tekniska Universitet, Sweden
†Department of Electrical Engineering and Automation, Aalto University, Espoo, Finland
Email: marco.romanato.3@studenti.unipd.it, sandeep.patil@ltu.se, dmitrii.drozdov@ltu.se, jerker.delsing@ltu.se, vyatkin@ieee.org

Abstract—This paper reports on integration of the Industrial Internet of Things (IIoT) service framework Arrowhead with automation systems programmed in the traditional and future automation languages. In particular it focuses on integrating the Arrowhead Datamanager system into the Eclipse 4DIAC IEC 61499 development environment and testing it on a physical system plant.

I. INTRODUCTION

Interoperability of automation systems with IoT and service platforms is an important problem in the Industry 4.0. The Arrowhead framework [1] is one such platform that aims at efficient support of the development, deployment and operation of interconnected, cooperative IoT systems. As shown in Fig. 1, it is composed of systems that provide and consume services. Different systems can be grouped together to form a Local Cloud (LC). Each LC needs to have three mandatory core systems: Service registry, Orchestrator and Authorization.

In Arrowhead, if a system X wants to request a service Y from another system it sends a request to the Orchestrator. It asks to the Service Register for the endpoint of the service Y and checks with the Authorization if the system X has the permission to demand for that service. In the positive reply case, the Orchestrator returns the information about service Y to the system X. In addition to the mandatory core systems, Arrowhead provides some supporting core systems as, for example, the Datamanager system that is what this paper focuses on. It allows to store and retrieve data to and from different systems and services.

II. CONNECTORS

There are two different ways to build an interface to Arrowhead services:
- Develop a service-interface function block (SIFB) in 4DIAC, using the C++ programming language.
- Develop the connector as a composition of library FBs from the existing 4Diac library.

The first method results in more compact executable code and most likely somewhat faster execution, but requires knowledge of the runtime internal organization to write SIFB. The learning curve for this approach is quite stiff and also such native SIFB requires extra work on porting (and testing) to newer versions of runtime. On the other hand, using library function blocks result in fully portable code and does not require additional knowledge in C++ and internal runtime
structure, though, execution will be slower, since even basic string operations invoke FB instances.

For this study we developed a set of composite function blocks (using the second approach) that allow to integrate Arrowhead Datamanager service "read" and "write" requests with 4diac applications.

Fig. 2 shows an internal FB network of a main communication CFB wrapper CommunicationFB. Its inputs are: data to be sent, the action to do and the endpoint to exchange data. The endpoint is a text string in the following format: IP+";"+PORT+"/datamanager/historian/"+Sys+"/Service", where "IP" and "PORT" specify an address of the Arrowhead Datamanager endpoint and "Service" and "Sys" identify the service of a system. Two possible actions are: PUT to write data and GET to read data. For the GET request, the FB outputs some Response data.

The CommunicationFB is composed of library FBs for string concatenation and conversion and CLIENT_1_2 FB for TCP communication. First, the Endpoint and Action inputs are concatenated as "http"+"Endpoint+Action+"/" to be used as ID input of the CLIENT_1_2 FB, and then the InputData is sent to a web service. The CommunicationFB serves as a base for two other FBs: Send{Type}ToDataManage and OutputHistorian{Type} which are the main connectors between IEC 61499 application and Arrowhead Datamanager service.

Send{Type}ToDataManage. It is the FB for writing data into the Datamanager. In Fig. 3 it is possible to observe the function block composition.

System and Service inputs are used by the InputHistorian{Type} FB, that concatenates them into a resulting string and returns data and endpoint in the correct format for the CommunicationFB. The required data structure is shown in Listing 1. When the Send{Type}ToDataManage FB is triggered, it also adds a timestamp to the data before sending it. Inside Send{Type}ToDataManage FB there is a dedicated FB that provides the current time based on the data and time of the PLC.

Listing 1: Data structure saved inside historian datamanager.

```
[ { "bn": "ServiceName", "bt": CurrentTime, "bu": "DataType", "bver": 1, "n": "ServiceName", "vs": "Data" } ]
```

OutputHistorian{Type}. It is used to get information from the Datamanager and its internal composition is shown in Fig. 4. It receives System and Service strings as inputs and requests a corresponding endpoint. As different from the Send{Type}ToDataManage, the InputHistorian{Type} FB is only used to get the right endpoint.

The output information of the CommunicationFB is a string and it is in the same format of Listing 1. That output string is elaborated by the GetDataFromDatamanagerResponse{Type} that extracts the value of the parameter "Data". In Fig. 5 it is shown that it is composed by BetweenStrings FB, that is given a string and an initial and final marker as inputs, and, returns the string portion between the markers.

BetweenStrings FB has been created using FBs from the IEC 61131-3/charString library of 4DIAC as shown in Fig. 6 to implement string processing.
Figure 6: Composition of the BetweenStrings FB.

In more details, the F_FIND and the F_MID are used at first to find the position of the start marker and take the string from the next character. Then, to the remaining string, the F_FIND is used again to find the position of the second marker, and then the F_DELETE FB to delete the characters after the second marker. Finally, the F_LEFT is used to avoid the presence of the space character at the end of the extracted string.

The {Type} of the previews blocks can be "String" or "Int". The only difference between them is where the variable related to the data value is stored. In the string case the sent data are like in list 1, while in the integer case, the Data is assigned to the "v" variable instead of the "vs".

III. CASE STUDY

The system used to validate the solution developed in this paper is composed by two conveyor belts, CB1 and CB2, that are driven by two IceBlocks nano-PLC controllers [7] (further referred as IceBlock101 and IceBlock103) (Fig 7). Each conveyor has start and end position sensors, and an actuator to move the belt. They are respectively connected to the inputs and output of the IceBlocks. IceBlocks communicate with each other and with the Arrowhead LC through a wireless network. The use of wireless communication in the automation sectors has increased greatly in recent years [8]. That is due to the fact that wireless technology is very cheap and it provides a high degree of deployment flexibility compared with the cables and that fit very well with the IEC 61499 standard. However its application in environments with strong radio noise could be a problem. But as tested in [9] there are some wireless standards that still allow one to have an high transmission efficiency.

The system network is composed by:

- a router, that allows to connect all the other devices.
- a computer that runs the Arrowhead Datamanager and the 4DIAC-ide application.
- two IceBlocks that run 4DIAC-forte and drive the respective conveyor belt.

In order to test the communication with the Arrowhead Data Manager, the 4DIAC application shown in Fig. 8 is used for both IceBlocks. The two applications differ slightly only for the function blocks IceBlock1 that is related to the control algorithm.

With the purpose of sharing some variables between the two IceBlocks, inside the Arrowhead Datamanager the following systems with related services have been created:

- TestAppIceBlockS1, with the service L2: It is related to the IceBlock103, that updates the service value. The other IceBlock instead requests the value of L2 to use it as input.
- TestAppIceBlockS2, with the service L3: It is related to the IceBlock101, that updates the service value. The other IceBlock instead requests the value of L3 to use it as input.
- TestAppIceBlock, with the service Last: Both the IceBlocks read and write on it. It is used to "share" a variable.

The application is used to cyclically move an object from the beginning of one conveyor belt to the end of the other conveyor belt and back to the beginning of the first one. More in detail, when an object activates the sensor L1 the conveyor belt CB2 starts moving forward and when the object reached the L2 sensor, the IceBlock103 updates the variable Last inside the datamanager; the other conveyor start moving in the same direction when Iceblock101 reads from Arrowhead that Last=1. When sensor L3 detects the object, Iceblock101 updates the variable Last on Arrowhead, and the previous conveyor is stopped. Finally when L4 is activated, Iceblock101 updates the Last variable and stop its conveyor belt. All these operation are executed in a way to obtain a "specular" result and move the object from sensor L4 to sensor L1.

The system in Fig. 8 is composed of the following FBs:

- E_CYCLE: it is used to send an event every 100ms that drives the following actions:
  - read the physical input (L1 and L2 for IceBlock103; L3 and L4 for IceBlock101).
  - write on the Arrowhead Datamanager of L2 and L3 variables respectively for IceBlock103 and IceBlock101.
  - read from Arrowhead Datamanager of Last and L2/L3 variables for Iceblock103/101.
• IX: it is used to read the input from the IceBlocks. The input \textit{PARAMS} of the FB is used to select the desired input of the IceBlock.

• QX: it is used to set the output of the IceBlocks. The input \textit{PARAMS} of the FB is used to select the desired output of the IceBlock.

• Send\{Type\}ToDataManage and OutputHistorian\{Type\}: as seen before are the FBs that allow the communication with Arrowhead.

• IceBlockTest2A and IceBlockTest2B: are used to control the movement of the conveyor belt based on the sensors value and the data form the Arrowhead Datamanager. The Moore state machines of the two FBs are shown in Fig. 9.

In both state machines the path $S_2 \Rightarrow S_3$ is performed when an object, is put in one of the ends of the conveyor belts (L1 for the IceBlock103, L4 for the IceBlock101). In this case, the IceBlock related to the activated sensor, makes the belt move in the direction of the other conveyor. Instead, the path $S_5 \Rightarrow S_6$ is performed when the object, transported by the other belt, reaches the end of it; the new belt is activated and moved in the same direction as the previous one. The LastOut output of states S3 and S4 is used to update the value of LAST inside the Datamanager. The two dotted lines show that the updated value of LAST can be used in state S1 of the other state machine.

IV. CONCLUSIONS

This paper is focused on the creation and the test of the connectors to communicate with the Arrowhead Datamanager, however, as discussed in the introduction, an application requesting an Arrowhead service shall first invoke such core services as ServiceRegistry, Orchestration and Authentication to make the communication in a correct and secure way. Connectors to these services are already implemented by 4DIAC community, for Arrowhead v4.1.2, however, the DataManager service was added only in 4.1.3, thus the case study has to be developed further to fully leverage Arrowhead framework. Future work can be aimed at porting the existing in 4DIAC function blocks for Arrowhead communication to work with Arrowhead 4.1.3 and extending the shown case study.

ACKNOWLEDGEMENTS

This work was supported, in part, by the Productive4.0 project, funded by the ECSEL joint undertaking, and by the 1-SWARM H2020 project.

REFERENCES

[1] P. Varga, F. Blomstedt, L. L. Ferreira, J. Eliasson, M. Johansson, J. Delsing, I. Martínez de Soria, Journal of Network and Computer Applications: “Making system of systems interoperable – The core components of the arrowhead framework”, 2017
[2] Arrowhead web page: https://www.arrowhead.eu/arrowheadframework/
[3] W. Dai, V. Vyatkin, J. Christensen, V. Dubinin, “Function Block Implementation of Service-Oriented Architecture: Case Study”, IEEE Conference on Industrial Informatics INDIN’2014, Porto- Allegre, Brazil, July 27-30, 2014
[4] W. Dai, V. Vyatkin, C. Pang, J. Peltoila, “Service-Oriented Distributed Control Software Design for Process Automation Systems”, IEEE Conf on Systems, Man and Cybernetics, San-Diego, 2014
[5] W. Dai, J. Christensen, V. Dubinin, V. Vyatkin, “Bridging Service-Oriented Architecture and IEC 61499 for Flexibility and Interoperability”, IEEE Trans on Industrial Informatics, 2015, 11(3)
[6] Jose Cabral, Kirill Dorofeev, Pál Varga, “Native OPC UA Handling and IEC 61499 PLCIntegration within the Arrowhead Framework”, 2020
[7] https://flexbridge.se/
[8] B. Hoffeld, D. Wieruch, T. Wirth, L. Thiele, S. Ali Ashraf, J. Huschke, I. Aktas, and J. Ansari, “Wireless Communication for Factory Automation: An Opportunity for LTE and 5G Systems”, 2016
[9] M. Žhan, Z. Pang, D. Dzung, M. Lavisotto, K. Yu, M. Xiao, “Towards High-Performance Wireless Control: 107 Packet Error Rate in Real Factory Environments”, IEEE Transactions on Industrial Informatics, 2019