Kajola, Paavo; Blech, Jan Olaf; Dwi Atmojo, Udayanto; Vyatkin, Valeriy

Dynamic Adapter Connections for IEC 61499

Published in:
Proceedings of 22nd IEEE International Conference on Industrial Technology, ICIT 2021

DOI:
10.1109/ICIT46573.2021.9453625

Published: 18/06/2021

Document Version
Peer-reviewed accepted author manuscript, also known as Final accepted manuscript or Post-print

Please cite the original version:
Kajola, P., Blech, J. O., Dwi Atmojo, U., & Vyatkin, V. (2021). Dynamic Adapter Connections for IEC 61499. In *Proceedings of 22nd IEEE International Conference on Industrial Technology, ICIT 2021* (pp. 1054-1059). Article 9453625 IEEE. https://doi.org/10.1109/ICIT46573.2021.9453625

Dynamic Adapter Connections for IEC 61499

Paavo Kajola*, Jan Olaf Blech*, Udayanto Dwi Atmojo*, Valeriy Vyatkin*†
*Department of Electrical Engineering and Automation, Aalto University, Espoo, Finland
†Department of Computer Science, Computer and Space Engineering, Lulea Tekniska Universitet, Sweden

Email: {paavo.kajola, jan.blech, udayanto.atmojo}@aalto.fi, vyatkin@ieee.org

***Abstract*—We present work on dynamic adapter connections for IEC 61499. Adapters in IEC 61499 simplify programs by encapsulating different event and data connections. Traditionally the plug and sockets are determined at design time of a system. In contrast, dynamic adapter connections do not statically link subsystems at design time, but allow to exchange interacting subsystems at runtime by retargeting plugs and sockets. This paper presents the dynamic adapter connection concept. In addition, we present an implementation concept based on OPC UA and an example.**

I. Introduction

Traditional automated production systems are mostly catering for mass quantity production. With mass quantity production in mind, the production lines in these systems are not subject to frequent changes during their lifetime. However, the flexible manufacturing paradigm is becoming more popular due to the increased business demand in customization of the product. A growing number of manufacturing companies which are adopting this paradigm are becoming more aware of the time and cost (efforts) needed to realize flexibility. It would be sensible to argue that having mechanisms that can automate and handle changes and reconfiguration can reduce efforts and costs.

In the industrial automation world, the IEC 61499 standard is gaining increased adoption over time and is generating more interest from automation solution vendors. The standard is an extension from the well-known IEC 61131-3, where control logic is implemented in the form of function blocks. A function block has multiple event and data interfaces, where the execution of the function block is triggered upon receiving event input (with any input data associated to the event, if any) and can generate event outputs (and also output data associated to it, if any) upon the end of the execution. Function blocks can have many event and data connections from and to the other function blocks, possibly causing the so called "spaghetti" connections.

The adapter interface concept was introduced in the IEC 61499 standard to make block diagrams tidier by means of encapsulating several data and event connections into one "thick cable", which can bundle connections between two function blocks going in both directions. This results in substitution of many connections by just one link between two function blocks. Initially, the use of adapter connections was considered only for composite function blocks, but the second edition of the standard suggested use of adapters also in basic function blocks.

The adapter concept defines an adapter provider ("plug") and an adapter acceptor ("socket"), where the event and data from the former could flow to the latter. Currently in the standard, the coupling between one plug and socket counterpart instance is set during design time and remains unchanged during runtime. However, it is foreseen that use cases that involves flexibility could require the coupling between one plug and its socket counterpart to change during runtime. Flexible manufacturing requires the reconfiguration of production facilities. An application should be able to deal with different devices such as sensors that are dynamically plugged into the system. These devices could also be occasionally be removed, e.g., do to maintenance. Therefore, we are looking into an extension for IEC 61499 supporting dynamic adapter connections and therewith flexible manufacturing infrastructures.

In this paper, we propose the concept of "dynamic adapter connections", an extension for the adapter concept in IEC 61499, which we design to enable plug and socket connections to change during runtime. We demonstrate a proof of concept implementation that automates the reconfiguration of plug and socket connections during runtime. We are particular aiming at dealing with unknown devices at development time of an application and allowing multiple readers to listen to a single writer on an adapter connection.

The paper is structured as follows. Some related works are presented in Section II. Section III describes our dynamic adapter connection approach. A proof of concept implementation of our approach is described in Section IV, complemented with an use case example that demonstrates how the implementation is used in 4DIAC environment in Section V.

II. Related Work

The innate capability of the underlying control software technology to support dynamicity in automated production is considered crucial by [14]. Adding and removing components at runtime is a key concept in many component frameworks in the IT world such as OSGi[1] in multi-agent systems paradigm such as JADE [7]), and also in system-level programming paradigm such as Service Oriented SystemJ (SOSJ) [8] and libDGALS [9]. Service oriented architecture concept defines that components should be loosely coupled to allow for flexible reconfiguration. In the past, we have proposed ideas to use more service oriented concepts in manufacturing [5] [12].

[1]https://www.osgi.org/

Here, we are considering the OPC UA technology as part of our dynamic adapter implementation.

Dynamic reconfiguration in the realm of IEC 61499 has been studied at multiple levels of abstraction including [1] and [3]. [13] presents some requirements for dynamic interface model in IEC 61499, but hasn't proposed an approach that demonstrates the idea. To our knowledge, we are the first to propose an approach to regard dynamicity for IEC 61499 adapter.

The adapter mechanism was found appealing by the application developers and has lead to design patterns such as "one line engineering" used in automation of material handling, manufacturing and process control systems [10], [11]. Some industry-driven standardisation bodies, such as Open Process Automation Forum (OPAF) [2] rely on the adapter mechanism for standardisation of domain-oriented design practices.

Our implementation relies on IEC 61499 and OPC UA, existing work comprises [4]. Furthermore, Service Oriented Architecture concepts have been studied in the IEC 61499 context, see, e.g., [2]. In addition, work on a service bus for IEC 61499 is presented in [6].

III. Dynamic Adapter Connection Concept

The conceptual idea of making adapter connections dynamic is shown in Figure 1. The figure compares the traditional approach (left side) of determining connections at design time with the dynamic reconfiguration possibility proposed in this paper (right side). In the original approach the connection is specified at design time. Once the resulting program has been uploaded to the devices it cannot be changed without uploading a new program. In our approach no new program needs to be uploaded in order to switch the connections. The connection controller block serves as an interface for the system to request connection switches.

While it is possible to create systems that switch adapter connections between static adapters (by connecting all adapter candidates to a "switchbox" FB that decides which input it should relay to the output), in practice it usually becomes unmanageable as the number of potential connections increases.

We propose new design artefacts referred to as dynamic plugs/dynamic sockets, and dynamic connectors. The dynamic plugs and sockets may be linked with each other at runtime to change the adapter connection. The dynamic plug and socket transfer the information encompassed by the adapter connection from one end to the other. Connections can be created and destroyed by sending a request via the connection controller.

IV. A Dynamic Adapter Implementation Based on 4DIAC and OPC UA

To demonstrate the concept of dynamic adapters, a proof of concept implementation is developed. The 4DIAC development tool[3] becomes our platform of choice due to open source license, good community of users and technical support, and frequent releases. Being an open source project, the tool has a rather extensive function block library that allows developers to focus on the application. We prototype the dynamic adapter connections using service interface function blocks (SIFB). In this prototype implementation, the static adapter connection between two function blocks is replaced by a dynamic connection implemented by using dedicated SIFBs that trigger the use of OPC UA to exchange data and events. Our implementation also considers the use of the OPC UA standard, as it has become a de facto communication standard in the Industry 4.0 context. It allows developers to take advantage of extensive information and data modelling. In this case it allows to model the adapter interface quite well.

Figure 2 shows the implementation of our dynamic adapter concept using OPC UA. Dynamic plugs and sockets communicate via OPC UA. The DYNACON_Plug and DYNACON_Socket function blocks use instances of OPC UA client in their implementation. The latter exchange the information via OPC UA server that runs in a separate process independent of any IEC 61499 devices. OPC UA provides a hierarchical data structure, which is well suited for representing adapters and their contents. Clients are expected to know the address and port of the server. When initialized, each endpoint connects to the OPC UA server and requests a set of variables to be created. These variables represent the adapter connection and its contents. An endpoint will only request to create its output values. For example, for a "one-way" adapter connection, where values written at the plug side are read from the socket side, only the plug will create variables on the server. The node creation process is currently handled by the server. The endpoint calls a method on the server and passes the names, types and number of variables as arguments. The server then creates these variables. Variables on the server are structured in the following manner. Each adapter has a node representing the adapter itself. The node has a numeric ID, given at design time by the user. Under the main adapter node are nodes for each of the event and data connections for the endpoint's half of the adapter interface. An endpoint will only write to its own set of variables. Adapters will read input variable values from their linked pair's nodes. For every endpoint, two special variables called "target" and "wakeup" are also created under the main node. "target" contains the ID of this endpoint's linked pair. "wakeup" is a generic event flag which signals that content in the adapter has changed.

Figure 3 presents a screenshot showing the data structure used by an adapter using an OPC UA inspection tool. Data and event nodes on the server are identified by unique string IDs, which consist of the variable name prepended by the numeric ID of the adapter to which they belong. In Figure 3, the "REQ" event variable under adapter 1 (Adp1) would have the string ID "1REQ". With this scheme it is not necessary to browse through the node tree to find variables. A client can directly access the variable node by using its node ID. ID 0 is reserved, and should not be allocated to an endpoint. Any endpoint with target ID 0 is considered unlinked.

Endpoint FBs are woken up periodically to poll the server

[2]https://www.opengroup.org/forum/open-process-automation-forum

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

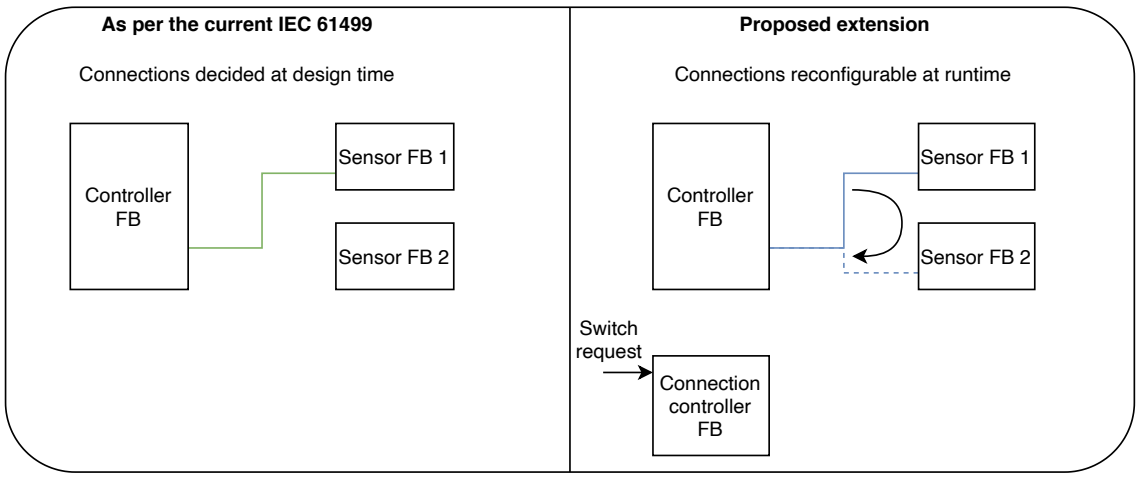

Fig. 1. Dynamic adapter connection concept.

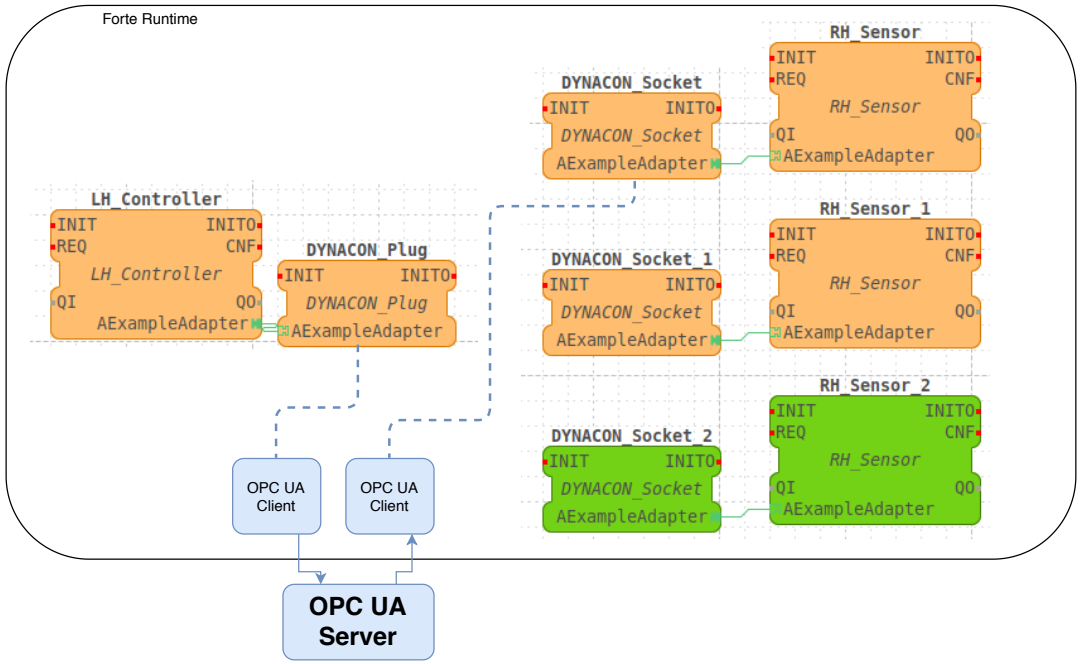

Fig. 2. Dynamic adapter connection implementation using OPC UA.

for new data to read. The reading process comprises several steps. First, an endpoint accesses its own nodes and reads the target variable. This gives it the ID of its pair. Next, the endpoint must read its pair's target variable to ensure a matched link. If the adapters' target variables point to something else than each other, the link is mismatched and the reading operation is aborted. After ensuring a two-way link, the endpoint checks whether the event flag for its pair is up. If the event flag is up, the endpoint may read its pair's nodes. Before reading the data, the endpoint will lower the event flag of its pair. Writing is simpler: to write, the endpoint only has to update its own data and event variables.

The connection controller SIFB enables the system to create and break links. It takes the requested link endpoints as input and activates upon reception of an event. The junction accesses both endpoints' target nodes on the server and sets the value to point to the pair's ID. It will send a confirmation event upon success, and a failure event if forming the connection fails. These events can be used by the rest of the system to react accordingly. Multiple connection controllers can exist simultaneously, but there are currently no measures to prevent the controllers from accessing the same endpoints simultaneously. It is up to the user to employ the controllers in a manner that ensures safe operation.

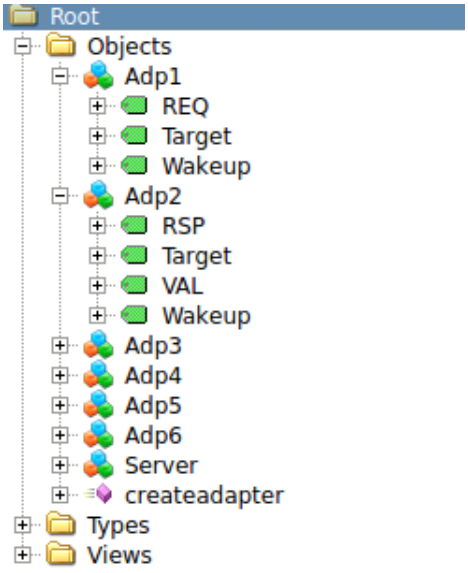

Fig. 3. Example Adapter Content viewed by an OPC UA viewer.

V. Step-by-step Example

Figure 4 shows an example system with dynamic adapter connections. The controller on the left calculates an average of the values of the virtual sensors on the right. The example uses simple placeholders for actual sensors. Each block outputs a constant value. The controller connects to each of the sensors going through the sensors one by one, reading their output value. After all the sensors have been visited, the controller will calculate the average and output it. The system can calculate the average for an arbitrary, changing number of sensors. Sensors may be activated and deactivated by spawning and killing instances of the 4DIAC runtime environment: Forte. The sensors in the example have each been mapped to different devices.

The data and events for the average calculation are contained in an adapter. The adapter connection is changed dynamically to allow the controller to connect to multiple FBs in sequential order. The dynamic adapter endpoints in this example are named DYNACON_Plug and DYNACON_Socket. These SIFBs represent the dynamic adapter connection. DYNACON_Plug and DYNACON_Socket communicate via OPC UA to transfer data from one adapter to another. Dynamic adapter connections between dynamic plugs and sockets may be established or broken by using the DYNACON_Junction SIFB. It is the connection controller of this system. All endpoints subscribe to Forte's timer handler to receive cyclic timer events. Each FB is activated by an external event sent by the timer handler. Upon activation, each FB checks if there is data to read on the server. The FBs receive a timer event every 500 milliseconds. The new SIFB types are created in 4DIAC. Their interface is defined in the same manner as other block types. The dynamic plug and socket FB type have initialization events INIT and INITO. They also have a data input called PLUG_ID and SOCKET_ID respectively. The interface of the dynamic plug contains an adapter socket, and the interface of the dynamic socket contains an adapter plug. The Junction FB interface contains events for initialization and updating the connection. It outputs one of two events after trying to link two endpoints. On success, it will send the CNF event. The FAIL event will be sent if connecting fails for any reason. The adapter itself consists of four events and one floating point value. Figure 5 shows the plug side adapter connector. The REQ and RSP events are used to send requests to the sensor and receive responses from it. NOREPLY_R and NOREPLY_L are not used by the example system, but exist for debugging and error handling purposes.

After creating the blocks in 4DIAC, they are exported as C++ source files by the 4DIAC exporter. The source code contains an interface specification and the logic of the FB. Exported SIFBs only contain the interface specification and no default logic. It must be implemented by editing the source file directly.

Figure 6 shows the class structure of the example system. By default, SIFBs inherit the basic functionality from the Forte FB base class CFunctionBlock. It contains generic functionality common to all FB types, such as common interface specification, event reception, and monitoring. The Forte basic, composite and event source FB classes are children of CFunctionBlock. The dynamic plug and socket blocks are changed to inherit from CEventSourceFB instead. This is required when an FB wants to subscribe to the Forte timer handler and receive timed events. The dynamic plug and dynamic socket classes also inherit from class DYNACON_Generic. This is a class made for this example that contains the necessities for communicating with the OPC UA server. These include an instance of the OPC UA client, a description of the adapter's contents and the routines to read from and write to the adapter nodes.

Each of the new FB types that communicate with the OPC UA server (DYNACON_Plug_thermo, DYNACON_Socket_thermo, DYNACON_Junction) connect to the server when they are initialized, and maintain the connection until the FB's destructor is called.

The example system contains the calculator composite block (Avg_calculator), five virtual / simulated sensors (Thermo), and a dynamic plug or socket FB for each. The average calculator contains data inputs for average recalculation period and ID range lower (inclusive) and upper (exclusive) bound. Figure 7 shows the network of function blocks inside the controller FB.

Acalc is a basic function block tasked with calculating the average value. It starts the calculation when it receives the "T" event. Acalc keeps track internally of where it is in the ID range. For each ID in the range, it will send the Junction block a request to connect the sensor with that ID to the average calculator. If forging the connection succeeds, the junction block will send a REQ event into the adapter line. The sensor generates an output value when it receives the REQ event. It will update its value (in this example, a constant) and send a RSP event as response through the adapter. Back

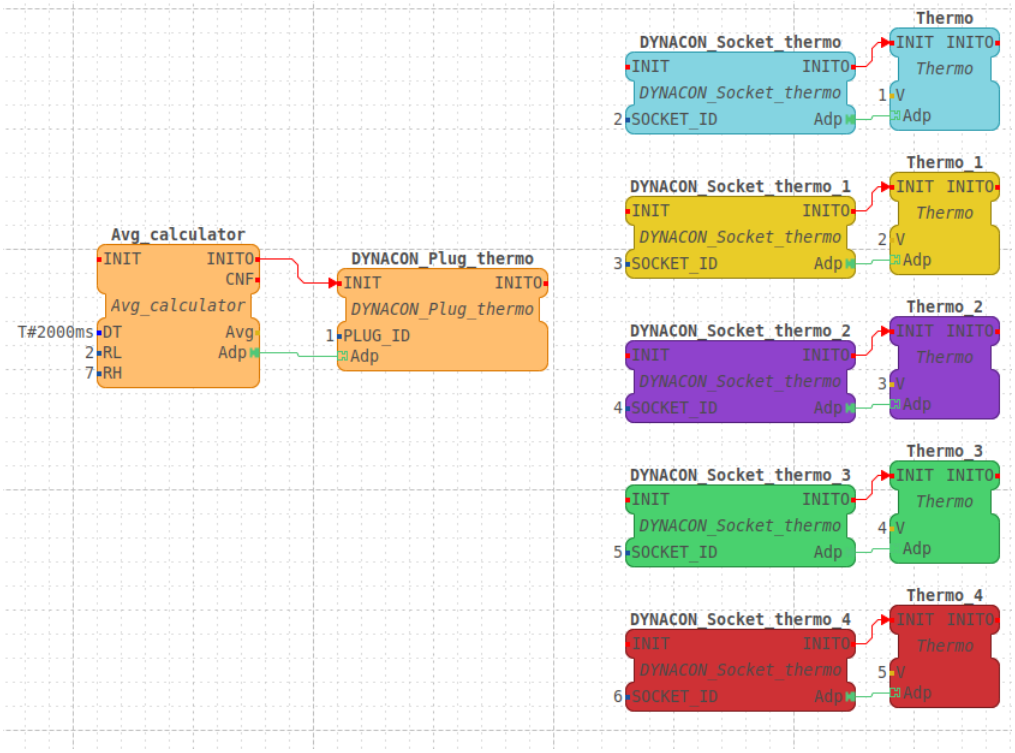

Fig. 4. Average calculator application.

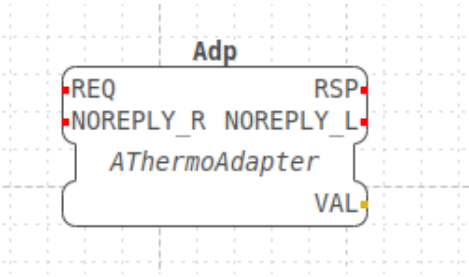

Fig. 5. The adapter plug.

inside Avg_calculator, the RSP event is connected to the ”R” event of Acalc, which tells it to proceed to the next ID in the range, and repeat the above steps. If the junction fails to connect the sensor to the average calculator for any reason, e.g. the sensor does not exist or is offline, it will send a FAIL event. This event, instead of going to the adapter, goes directly back to Acalc, and tells it to skip this sensor. When Acalc reaches the final ID in its range, it will calculate the average and write it to its output. It will also send the OUT event to signal that a new average value has been calculated. The OUT signal is also connected to a delay block, which will, after a delay, send Acalc another ”T” event, thus restarting the average calculation. In this example the average is recalculated every 2 seconds.

VI. Conclusion

We have presented an approach, implementation and example to dynamically ”rewire” adapter connections in IEC 61499. This supports flexible manufacturing concepts where some devices may change at runtime or are not known at design time of a system.

Future work comprises the extension of the described concept to other environments and the creation of associated services. The challenges addressed in this paper are not only relevant for the industrial automation domain and IEC 61499, but the concept could be ported to the MATLAB/Simulink tool suite which has some popularity in the embedded community and in automotive systems in particular. The designs presented in this paper did not contain methods to search for endpoints. In larger systems it is unlikely that the user is able to predict and plan all device identifiers at design time. An ability to find endpoints by description or associated keywords would greatly improve viability. Each endpoint should have brief descriptive tags, e.g. ”sensor” and ”temperature”, which could be used to identify the type of device and its function.

VII. Acknowledgements

This work was sponsored, in part, by the Reboot IoT project (funded by Business Finland, https://rebootiotfactory.fi/), EIT Manufacturing and by the HORIZON2020 project 1-SWARM co-Funded by the European Commission (grant agreement: 871743). The authors are grateful to Hidenori Sawahara of Yokogawa Electric for inspiring us with the challenge of dynamic adapter connections.

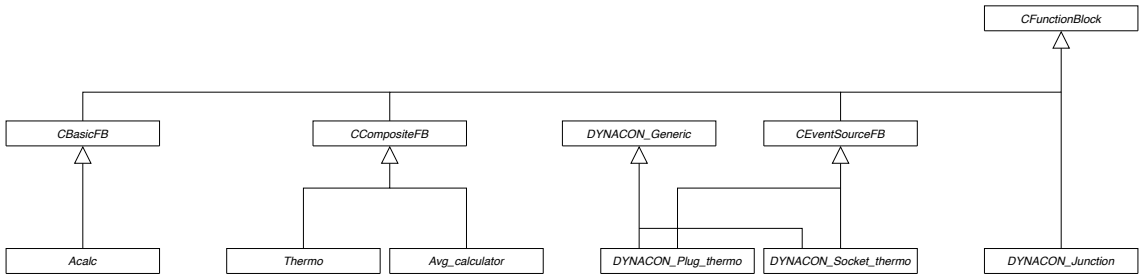

Fig. 6. Class inheritance of Forte FBs.

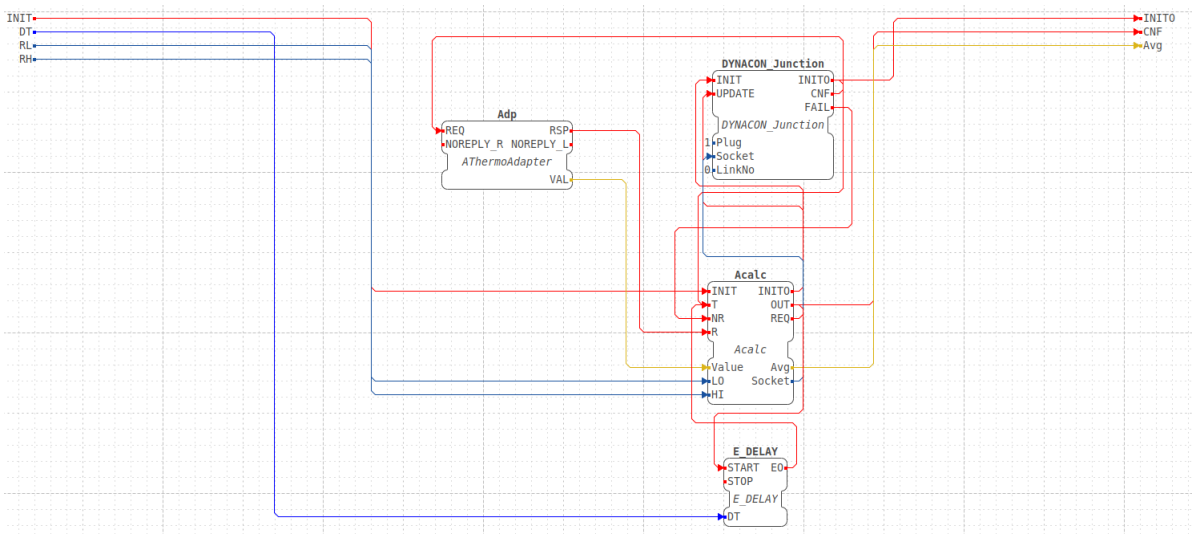

Fig. 7. Average calculator FB internals.

REFERENCES

[1] A. Zoitl, C. Sunder, & I. Terzic. "Dynamic reconfiguration of distributed control applications with reconfiguration services based on IEC 61499." IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications (DIS'06). IEEE, 2006.

[2] W. Dai, V. Vyatkin, J. H. Christensen, & V. N. Dubinin (2015). Bridging service-oriented architecture and IEC 61499 for flexibility and interoperability. IEEE Transactions on Industrial Informatics, 11(3), 771-781.

[3] I. Batchkova, G. Popov, H. Karamishev, & G. Stambolov (2013). Dynamic reconfigurability of control systems using IEC 61499 standard. IFAC Proceedings Volumes, 46(8), 256-261.

[4] T. Terzimehic, M. Wenger, A. Zoitl, A. Bayha, K. Becker, T. Müller, & H. Schauerte (2017, September). Towards an industry 4.0 compliant control software architecture using IEC 61499 & OPC UA. In 2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA) (pp. 1-4). IEEE.

[5] V. Kuliaev, U.D. Atmojo, S. Sierla, J.O. Blech, & V. Vyatkin, "Towards Product Centric Manufacturing: From Digital Twins to Product Assembly", 17th International Conference on Industrial Informatics (IEEE-INDIN 2019) July 22-25, 2019 Helsinki, Finland.

[6] V. Ashiwal, A. Zoitl & M. Konnerth, "A Service Bus Concept for Modular and Adaptable PLC-Software," 2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Vienna, Austria, 2020, pp. 22-29, doi: 10.1109/ETFA46521.2020.9211908.

[7] F. Bellifemine, F. Bergenti, G. Caire, & A. Poggi (2005). "JADE—a java agent development framework". In Multi-agent programming (pp. 125-147). Springer, Boston, MA.

[8] U. D. Atmojo, Z. Salcic, K. I. Wang & V. Vyatkin, "A Service-Oriented Programming Approach for Dynamic Distributed Manufacturing Systems," in IEEE Transactions on Industrial Informatics, vol. 16, no. 1, pp. 151-160, Jan. 2020, doi: 10.1109/TII.2019.2919153.

[9] W. Sun, Z. Salcic, A. Girault & A. Malik, "libDGALS: A library-based approach to design dynamic GALS systems," Proceedings of the 9th IEEE International Symposium on Industrial Embedded Systems (SIES 2014), Pisa, 2014, pp. 104-111, doi: 10.1109/SIES.2014.6871194.

[10] G. Kollegger, & A. Kopitar, "Flexible and Reusable Industrial Control Application", in Distributed Control Applications: Guidelines, Design Patterns, and Application Examples with the IEC 61499, pp. 245-273, 2016, CRC Press

[11] P. Jhunjhunwala, U. D. Atmojo & V. Vyatkin, "Towards Implementation of Interoperable Smart Sensor Services in IEC 61499 for Process Automation," 2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Vienna, Austria, 2020, pp. 1409-1412, doi: 10.1109/ETFA46521.2020.9211925.

[12] U. D. Atmojo, J. O. Blech, S. Sierla & V. Vyatkin, "Service-based Architecture with Product-centric Control in a Production Island-based Agile Factory," 2019 IEEE International Conference on Industrial Internet (ICII), Orlando, FL, USA, 2019, pp. 305-306, doi: 10.1109/ICII.2019.00060.

[13] B. Wiesmayr, & A. Zoitl, "Requirements for a dynamic interface model of IEC 61499 Function Blocks," 2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Vienna, Austria, 2020, pp. 1069-1072, doi: 10.1109/ETFA46521.2020.9212107.

[14] B. Vogel-Heuser, A. Fay, I. Schaefer, & M. Tichy, "Evolution of software in automated production systems: Challenges and research directions,"Journal of Systems and Software, vol. 110, pp. 54–84, 2015