Jhunjhunwala, Pranay; Blech, Jan Olaf; Zoitl, Alois; Atmojo, Udayanto Dwi; Vyatkin, Valeriy

A Design Pattern for Monitoring Adapter Connections in IEC 61499

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

DOI:
10.1109/ICIT46573.2021.9453685

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:
Jhunjhunwala, P., Blech, J. O., Zoitl, A., Atmojo, U. D., & Vyatkin, V. (2021). A Design Pattern for Monitoring Adapter Connections in IEC 61499. In *Proceedings of 22nd IEEE International Conference on Industrial Technology, ICIT 2021* (pp. 967-972). Article 9453685 IEEE. https://doi.org/10.1109/ICIT46573.2021.9453685

A Design Pattern for Monitoring Adapter Connections in IEC 61499

Pranay Jhunjhunwala*, Jan Olaf Blech*, Alois Zoitl‡, 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
‡ Johannes Kepler University Linz, Linz, Austria

Email: {pranay.jhunjhunwala, jan.blech}@aalto.fi, alois.zoitl@jku.at, udayanto.atmojo@aalto.fi, vyatkin@ieee.org

***Abstract*—Today's software developments are frequently structured into different components with well defined interfaces. IEC 61499 comes with well defined interface mechanisms such as adapters that group data and event exchange between different subsystems. Compliance with interface specifications can be monitored at run-time. In this paper we present a design pattern to monitor adapters thereby observing whether communication specifications are fulfilled. We present an example demonstrating monitoring of a handshaking mechanism used between control application components.**

***Index Terms*—Component automation architecture, IEC 61499, standard interfaces, monitors.**

I. Introduction

Software development in industrial automation increasingly relies on software components such as communication stacks, algorithms and specialized drivers. These components are typically developed and maintained by different teams such as specific vendors, or teams from different departments within the same company. A control engineer relies on these components to build an application. Public access to the source code of these components is not always provided, e.g., due to Intellectual Property Rights. For this reason it is hard, sometimes impossible, to test and analyse the exact internal behavior of these components ("black-box" components). Nevertheless, an integrated application relies on specified properties of these black-box components. Since a thorough verification and exhaustive testing is not always possible before deployment of an application, it can be advisable to monitor the behavior of components during run-time. This is particularly true in cases where components can be updated independently of an application, such as the installation of a new driver with a slightly different behavior. In particular communication interfaces and in some cases resource consumption of black-box components can be monitored.

In the industrial automation realm, IEC 61499 represents a modern component-based architecture and is frequently used for the specification of control applications. A core feature of component-based architectures are their well defined interfaces and their interactions. IEC 61499 defines for that the adapter interface and adapter connection concept, which allows to define the full interface in a single entity (i.e., events and data) between two specific components. This overcomes the separation of events and data used typically in IEC 61499 applications. Adapters simplify connections by structuring and aggregating component interconnections. Thus, for studying IEC 61499 component interactions, adapter connections are at the core of the investigation. Therefore, in this paper, we are focusing on interface monitors for IEC 61499-based software components that communicate via adapters with each other. We are particularly interested in introducing a design pattern for the monitoring of IEC 61499 adapters. Our monitors are based on formal specification mechanisms such as state machines and are able to detect deviations from the expected behavior. In that case the application is notified via an event and can take countermeasures such as transitioning into a safe state.

Some design patterns have already been proposed in relation to adapters (e.g. [1], [2]). Work on verification of white-box components interaction is presented in [3]. Run-time verification (e.g. [4], [5]) is a technology that allows for checking specifications at run-time of a system. Our monitoring patterns follow this idea. Observer-based verification for IEC 61499 has been proposed in [6]. Work on monitoring of IEC 61499 specifications using formal specifications is proposed in [7]. Based on that [8] presents a remote monitoring infrastructure, while [9] focuses on specification formalisms for monitoring and an integration into the Eclipse environment. A so-called sniffer pattern is introduced in [10]. An investigation on the correctness of run-time monitors themselves has been conducted in [11].

Monitoring as one technique to identify faults at runtime for industrial systems is also listed in a survey [12]. The work in [13] shows an example where complex communication protocols (called channels) originally specified in a language called SystemJ [14] were adopted and implemented in IEC 61499. While SystemJ software components are associated with behaviors that naturally have underlying state machines, mapping the SystemJ channel specifications into state machines are challenging, especially due to the lack of software tool to facilitate the mapping. Thus, there is no guarantee that the current IEC 61499 channel implementation, which are defined in state machines-based ECC, would behave according to the original specifications. Here, monitors could be used, e.g., during the testing phase of the development or certifications,

that the implementation adheres to the specifications and to detect any violations, incorrect behaviors.

The rest of this paper is structured as follows: Section II describes the proposed monitoring pattern. Section III considers implementation options for the proposed monitoring pattern in IEC 61499. Section IV introduces an illustrative example to which the proposed monitoring pattern is applied in Section V. Section VI concludes the paper and presents potential next steps.

II. The Monitoring Pattern

The general setting for this work is that we have a set of interacting components as shown in Fig. 1. An arbitrary number (n) of components are connected with components (m) using adapter connections. We assume that we have at least one consistency condition that involves all adapter connections. The goal is to have a monitor which will observe all adapter connections and determine whether this consistency condition is fulfilled. If it is violated a notification to the application or an higher level system shall be generated allowing to perform certain countermeasures.

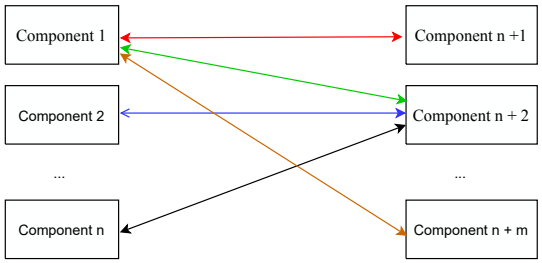


Fig. 1: Generic component interaction scenario which shall be monitored.

In previous work [7], [8] we investigated such monitors by enhancing the IEC 61499 execution environment provided by the Eclipse 4diac project[1]. While it showed great potential, it required a deep change in the execution environment. Furthermore this approach did not allow an easy interaction between monitor and the IEC 61499 application and monitoring results where invisible to the application developer. Ref. [10] presented an approach for extracting common error handling code by introducing a design pattern which allowed to intercept adapter connections feedback free. To overcome the limitations of the previous monitoring approaches we propose to expand the adapter interception concept from [10] and to explicitly add the monitor components as part of the IEC 61499 application.

Fig. 2 shows the same situation after adding the proposed explicit monitor concept. The adapter connections to be monitored are split into two parts with the monitor in the middle. The monitor shall only observe the data and events that are being transferred over the split up adapter connections and may not alter it. The monitor can create an event "Monitoring Violation". This can be used to inform the IEC 61499 application about monitoring violations and based on that handle error recovery.

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

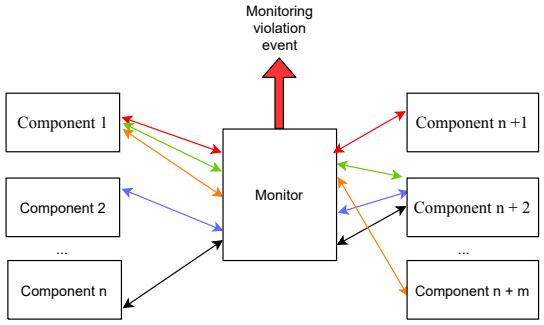


Fig. 2: A generic IEC 61499-based design pattern for monitoring component interactions at IEC 61499 application level.

For example, by potentially invoking other parts of the IEC 61499 application.

III. Implementation Options

In the IEC 61499 architecture, the interacting components, mentioned in the previous section, can be implemented using such design artefacts as *function blocks* (FBs) or *subapplications*. The monitor can also be implemented using the same kind of artefacts.

Fig. 3 presents an example of implementation of the proposed monitor pattern by means of IEC 61499 architecture, using adapter connections between FBs.

- The monitor FB/subapplication is "inserted" in the adapter connections between the interacting components (cf. Figs. 1 & 2).
- The monitor FB/subapplication has event and data outputs for informing on monitoring results (e.g., MonitoringViolation).
- Event and data inputs may allow the application to interact with the monitor, e.g., triggering modes switches, handle error conditions (not shown in Fig. 3).

A. Monitor Specification Options

The consistency condition to be monitored can be specified by means of various specification languages. Dependent on the type of the condition, such options as state machines, Petri nets, process algebras, etc., can be explored. While

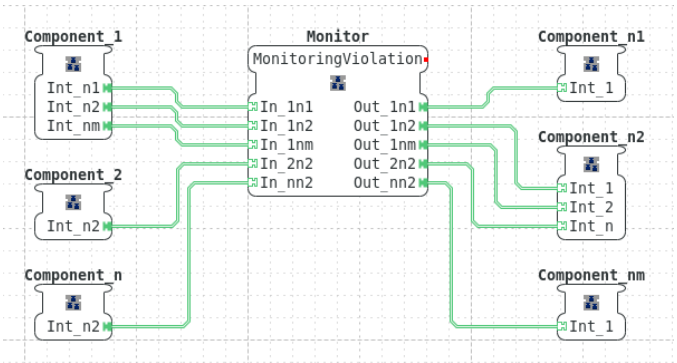


Fig. 3: IEC 61499 implementation of the proposed design pattern for monitoring multiple adapter connections.

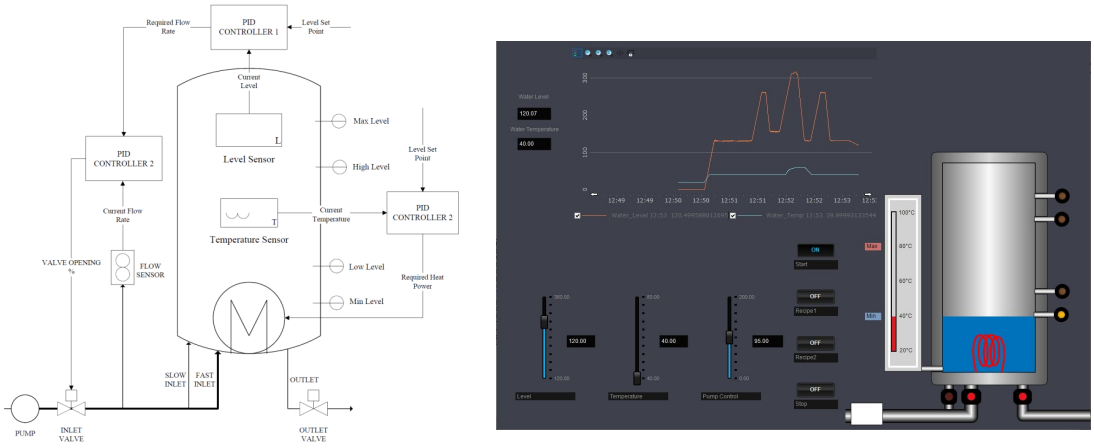


Fig. 4: Example system.

these specification means have the required expressiveness, most of them are not directly implemented as a part of IEC 61499 standard. In this first investigation we therefore want to investigate which IEC 61499 means are suitable for implementing monitoring functionality in the monitoring FB/subapplication (cf. Fig. 3). However the below presented approaches can always be wrapped in subapplications for encapsulation if needed.

The IEC 61499 option with the most efficiency would be to use Service Interface FBs (SIFBs). A SIFB defines only its interface to interact with the application via events, data, and adapter connections. Its internals, however, are defined with means outside of the scope of IEC 61499 and are vendor/run-time environment specific. This has the great advantage that the specification/implementation means best suited for monitoring could be utilized, while at the same time allow tight interaction with the monitored application. However, this is only a small improvement to the previous approaches [7], [8] as the results are very specific to a certain run-time environment.

The second IEC 61499 FB suitable would be the Basic FB (BFB). A BFB contains a state chart mechanism called Execution Control Charts (ECC). The ECC is very well suited for specifying the consistency conditions of a monitor. The drawback of BFBs is that the event and data flow which the monitoring FB is intercepting (cf. Fig. 3) has to be manually maintained so that the interaction between the components is not disturbed. That means in addition to the monitoring states and transitions dedicated states and transitions have to be added to the ECC which will ensure the event and data flow. This could result in very big and hard to maintain ECCs, and therefore this solution seems not appropriate.

As final option IEC 61499 offers Composite FBs (CFBs). CFBs encapsulate a FB network consisting of one or more FB instances and connections between contained FBs and the interface of the CFB. Adapter interfaces (i.e plugs and sockets) are represented as FBs in the internal FB network of a CFB. This has the great advantage that for maintaining the event and data flow of the intercepted adapter connection only the IOs of the plug and its according socket need to be connected. The monitoring functionality of a CFB need to be implemented by means of interaction of its constituent parts. An appropriate solution can be to use one or several BFBs as discussed in the previous paragraph. But the BFBs would not directly feature the plug and sockets of the intercepted connections but would be connected via event and data connections to the plug and socket representations in the CFB. With that a best of both worlds approach can be achieved: the BFB with its ECC is used for the monitor specification, the discrete representation of the plugs and sockets in the CFB is used for an easy maintenance of the event and data flow and for extracting the events and data for the monitoring BFB. This promising approach is further investigated in the rest of this paper.

B. Deployment Options

A further aspect of monitoring component interactions is where to perform the analysis. This can be either local or remote. Local has the advantage of fast and direct interaction with the application. Remote monitoring has the advantage that the data processing can be performed an a separate computing device (i.e. cloud or edge). This can reduce the impact of monitoring on the application execution especially when the monitoring checks get more complex or when data from any parts of the application shall be checked (e.g. [8]).

In our proposed approach both options can be implemented. In the local approach, the implementation options, discussed in the previous section, would just be deployed as a part of the monitor FB/subapplication. For remote deployments, the monitor FB/subapplication would contain a set of communication FBs which would forward all monitored data in an appropriate form to the computing device performing the analysis. Any feedback generated by the remote monitoring algorithm would be received again with communication FBs. Although both options are hidden within the monitoring FB/subapplication, they can be transparent for the application engineer. For the first investigation of our proposed monitoring approach we will focus on the local monitoring.

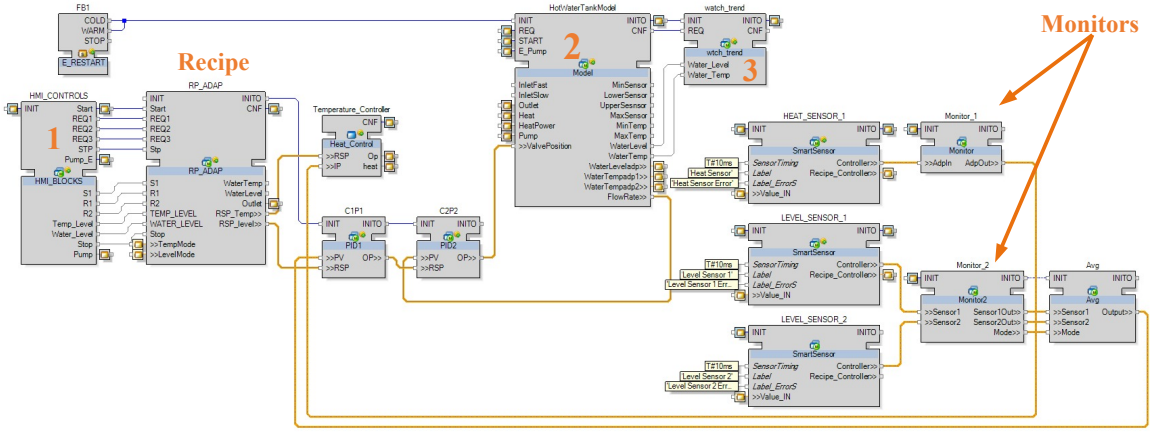


Fig. 5: Overview on the function block application with the component interactions to be monitored.

IV. Example

This section presents an example to demonstrate the feasibility of our monitoring pattern. As a case study, we use a single tank reactor system that consists of a vessel with a heating element, connected with pipelines to the source and sink of the liquid. The piping and instrumentation diagram of this tank reactor example is shown in Fig. 4 (left). The tank is supplied with liquid via two inlet pipes, referred to as slow inlet and fast inlet. The solution entering the inlet pipes is regulated by a pump installed at the incoming pipe. There is one outlet pipe which drains the tank as needed. The liquid in the tank is heated by a heating coil connected at the bottom of the tank and the temperatures can be seen on the indicator to the left of the tank.

The setup consists of various sensors to measure the level and temperature in the tank as seen in Fig. 4 and the communication between the level and temperature sensors with their respective controllers is being monitored using the presented design pattern and idea for monitors. Monitors used are shown in Fig. 5 and will be explained in the subsections below.

A. The Handshake Mechanism

A handshake mechanism has been developed to confirm the transmission of sensor values from the smart sensor to the controller. The mechanism has been developed as a part of the smart sensor and is a part of the duplex communication channel implemented using the IEC 61499 adapter technology. With the help of the handshake mechanism the smart sensor can send messages along with the values it is sending to the controller and then awaits a confirmation from the controller. Only upon receiving the confirmation via the handshake protocol does the smart sensor send the next or updated value. Sequence diagram in Fig. 6 depicts the desired operation of the handshake mechanism for two scenarios.

Scenario 1 'Full operation' of the sequence diagram depicts the desired or ideal mechanism of the handshake protocol wherein the sensor sends 'value XX', along with that 'Sent,1' string is appended. Once the controller successfully receives the value it sends out a string 'Rec,1' indicating to the smart

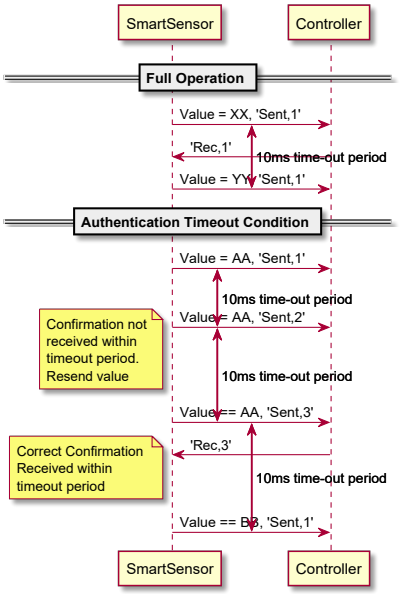


Fig. 6: Handshake Mechanism.

sensor that it received the value sent, after which the sensor sends the updated value. *Scenario 2 'Time out condition'* on the contrary explains the requirements when the confirmation is not received by the smart sensor within the set waiting time period. As shown in the sequence diagram the sensor sends 'value AA along' with the appended message 'Sent,1' but does not get a confirmation within the 10ms time-out period and hence resends 'value AA' but with an updated message 'Sent,2' indicating that it is sending the same value the second time. This cycle continues until the sensor receives a confirmation within the 10ms time-out period. Once the correct confirmation is received it sends the updated value.

V. Designed Monitors

A. Monitor 1: Handshake Mechanism Monitor

A monitor, as shown in Fig. 7a, has been developed to verify the handshake mechanism. The monitor FB is placed in between

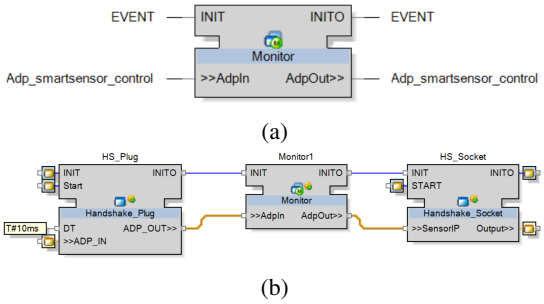


Fig. 7: Monitor 1 a) Interface and b) Connections.

the plug and socket of the handshake mechanism as depicted in Fig. 7b and works on top of the communication channel between the sensor and the controller, if the monitor finds an error with the protocol of the handshake mechanism it raises an alarm on the HMI notifying the issue in the implementation.

The monitor FB has been designed as a CFB in a pattern shown in Fig. 8, to not disturb the communication between the channel rather just take information from the channel and check it based on the state machine inside the MonitorFB1 basic FB. The adapter plug and sockets are directly connected i.e. enabling direct communication from the sensor to the controller and vice-versa.

Represented in Fig. 9 is the state machine designed for the monitor 1 operations. The monitor can be enabled using a button in the HMI which will take the state machine to the wait state, once the monitor receives a value from the smart sensor it will first check if the value is an old value or an new value, after which it moves to the corresponding state and checks if the appended string for the handshake mechanism is according to the desired specifications. If the incoming messages are correct the system then proceeds to check for the confirmation message sent by the receiver. This state machine runs in a continuous loop until a received message does not correspond to the expected message, in which case the state machine progresses to the error state and stays there until manual event is not generated. Along with that the monitor also raises an error flag on the HMI and informs where the error is and what the error is.

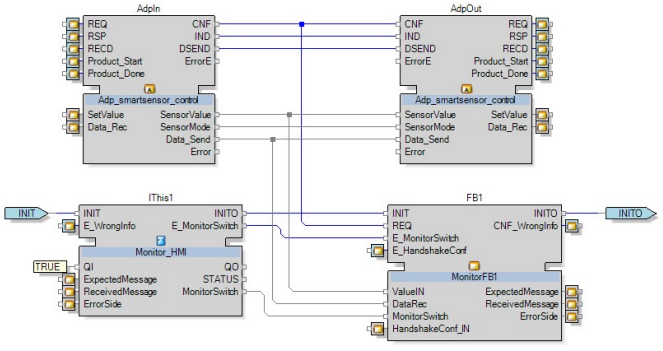


Fig. 8: Monitor 1 Architecture

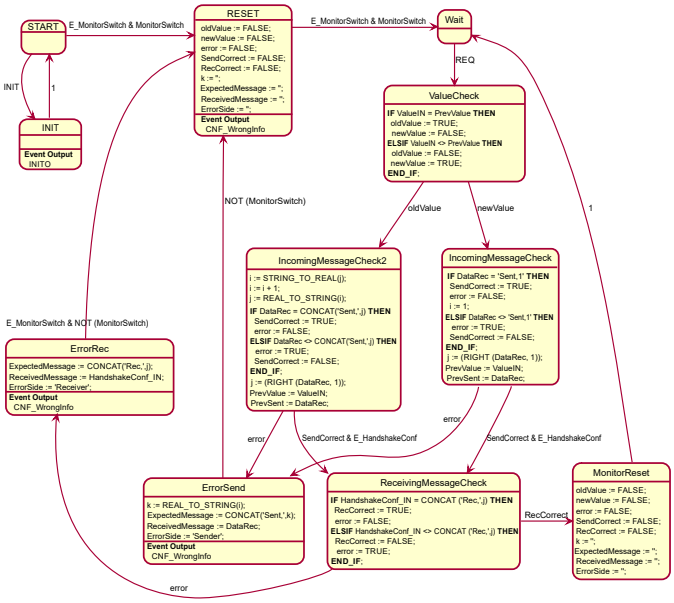


Fig. 9: Monitor 1 State Machine.

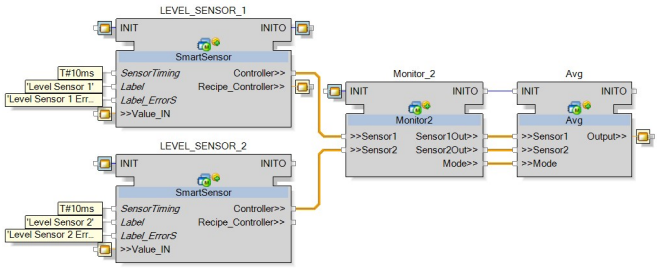
B. Monitor 2

A monitor was designed to analyze various subsystems and then based on the observations command a set of further subsystems to perform the desired operations. In our development and test case, shown in Fig. 10a, we used 2 heat sensors in the water process tank acting as the 2 subsystems to be monitored, which was then commanding the average FB to use the values from one of the sensor or both the sensors based on the received data.

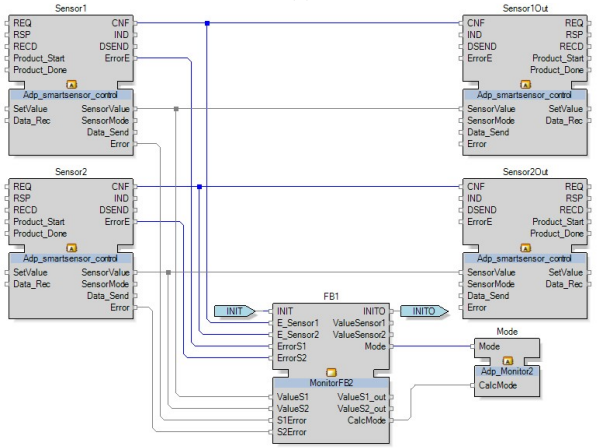
As shown in Fig. 10b, the monitor basic FB acts on top of the communication channel between the sensors and the controls and monitors the sensors for damage. The MonitorFB2 basic FB, is specified following the state machine in Fig. 10c, which has 3 states of output i.e. use sensor 1, use sensor 2 or use the average of both sensors. Designed to monitor the operation of various subsystems and the quality of the channel, the monitor continuously assesses the functioning of the individual subsystems i.e. their activity and functioning. If any of the subsystem gets damaged or stops functioning, the monitor is capable of raising an alarm regarding the same. To monitor the channel quality, the MonitorFB2 uses the feature of timestamps included in the design wherein it calculates the delay in which it is receiving values and then the subsystem with lower delay is recommended by the monitor.

VI. Conclusion and Outlook

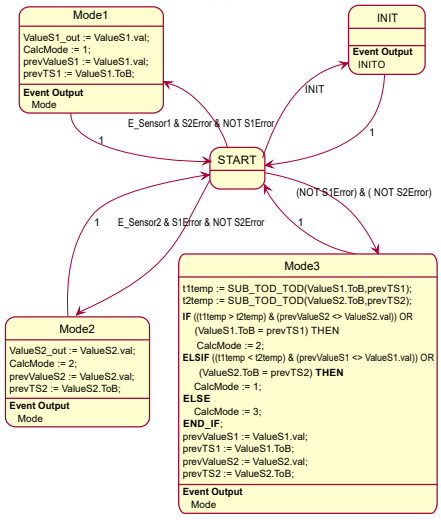
In this paper we presented a design pattern for monitoring of IEC 61499 adapter connections. Implementation options for the proposed design patterns are discussed and example implementations and applications are presented. Our pattern does not only allow the detection of deviations from a specification but provides a well defined way to communicate this deviation to the application. This can be used in a similar way as the well-known exception handling mechanism that is



(a)



(b)



(c)

Fig. 10: Monitor 2 a) Use Case, b) Connection, and c) State Machine

available in many higher-programming languages such as Java. Right now, the monitor specification and implementation relies completely on existing IEC 61499 specification options. This has the great advantage that the monitor is independent from run-time systems and vendors.

For future work, two major areas can be identified. At first, we are interested in investigating different specification languages for the monitors. These comprise regular expressions, Petri nets, automata, process algebras, temporal logic, and other formalisms that come with a well defined semantics. Automatic generation of monitors from these specifications is closely connected to this. In addition, we are interested in looking into distributed monitoring. When monitoring different adapters that are deployed on distributed devices, the communication overhead can be significant. Distributing a monitor over different devices could be a solution to reduce the communication overhead, especially if the properties to be monitored only have sporadic interdependencies.

VII. Acknowledgements

This work was sponsored, in part, by the EIT Manufacturing and by the H2020 project 1-SWARM co-funded by the European Commission (grant agreement: 871743).

References

[1] 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.

[2] D. Drozdov, U. D. Atmojo, C. Pang, S. Patil, M. I. Ali, A. Tenhunen, T. Oksanen, K. Cheremetiev, and V. Vyatkin, "Utilizing software design patterns in product-driven manufacturing system: a case study," in *International Workshop on Service Orientation in Holonic and Multi-Agent Manufacturing*, pp. 301–312, Springer, 2019.

[3] H. Prähofer and A. Zoitl, "Verification of hierarchical IEC 61499 component systems with behavioral event contracts," in *11th IEEE International Conference on Industrial Informatics*, pp. 578–585, IEEE, 07 2013.

[4] M. Leucker and C. Schallhart, "A brief account of runtime verification," *The Journal of Logic and Algebraic Programming*, vol. 78, no. 5, pp. 293–303, 2009.

[5] H. Barringer, A. Goldberg, K. Havelund, and K. Sen, "Rule-based runtime verification," in *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pp. 44–57, Springer, 2004.

[6] Z. E. Bhatti, R. Sinha, and P. S. Roop, "Observer based verification of IEC 61499 function blocks," in *2011 9th IEEE International Conference on Industrial Informatics*, pp. 609–614, IEEE, 2011.

[7] M. Wenger, A. Zoitl, and J. O. Blech, "Behavioral type-based monitoring for IEC 61499," in *20th IEEE Conference on Emerging Technologies & Factory Automation, ETFA 2015, Luxembourg, September 8-11, 2015*, pp. 1–8, IEEE, 2015.

[8] M. Wenger, A. Zoitl, J. O. Blech, and I. Peake, "Remote monitoring infrastructure for IEC 61499 based control software," in *8th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops, ICUMT 2016, Lisbon, Portugal, October 18-20, 2016*, pp. 369–374, IEEE, 2016.

[9] J. O. Blech, Y. Falcone, H. Rueß, and B. Schätz, "Behavioral specification based runtime monitors for OSGi services," in *Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change - 5th International Symposium, ISoLA 2012, Heraklion, Crete, Greece, October 15-18, 2012, Proceedings, Part I* (T. Margaria and B. Steffen, eds.), vol. 7609 of *Lecture Notes in Computer Science*, pp. 405–419, Springer, 2012.

[10] M. Steinegger, A. Zoitl, M. Fein, and G. Schitter, "Design patterns for separating fault handling from control code in discrete manufacturing systems," in *IECON 2013-39th Annual Conference of the IEEE Industrial Electronics Society*, pp. 4368–4373, IEEE, 2013.

[11] J. O. Blech, Y. Falcone, and K. Becker, "Towards certified runtime verification," in *International Conference on Formal Engineering Methods*, pp. 494–509, Springer, 2012.

[12] B. Dowdeswell, R. Sinha, and S. MacDonell, "Finding faults: A scoping study of fault diagnostics for industrial cyber-physical systems," *Journal of Systems and Software*, p. 110638, 2020.

[13] U. D. Atmojo and V. Vyatkin, "A design pattern for systems composed from intelligent mechatronic modules with wireless communication," in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 41–48, IEEE, 2019.

[14] A. Malik, Z. Salcic, P. S. Roop, and A. Girault, "Systemj: A GALS language for system level design," *Computer Languages, Systems & Structures*, vol. 36, no. 4, pp. 317–344, 2010.