Atmojo, Udayanto Dwi; Gulzar, Kashif; Vyatkin, Valeriy; Ma, Rongwei; Hopsu, Alexander; Makkonen, Henri; Korhonen, Atte; Phu, Long Tran

Distributed control architecture for dynamic reconfiguration

Published in:
Proceedings of the 1st IEEE Industrial Cyber-Physical Systems, ICPS 2018

DOI:
10.1109/ICPHYS.2018.8390791

Published: 15/06/2018

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

Please cite the original version:
Atmojo, U. D., Gulzar, K., Vyatkin, V., Ma, R., Hopsu, A., Makkonen, H., Korhonen, A., & Phu, L. T. (2018). Distributed control architecture for dynamic reconfiguration: Flexible assembly line case study. In *Proceedings of the 1st IEEE Industrial Cyber-Physical Systems, ICPS 2018* (pp. 690-695). IEEE.
https://doi.org/10.1109/ICPHYS.2018.8390791

Distributed Control Architecture For Dynamic Reconfiguration: Flexible Assembly Line Case Study

Udayanto Dwi Atmojo[1], Kashif Gulzar[1], Valeriy Vyatkin[1,2], Rongwei Ma[1], Alexander Hopsu[1], Henri Makkonen[1], Atte Korhonen[1], Long Tran Phu[1]

[1]Department of Electrical Engineering and Automation, Aalto University, Finland
[2]Department of Computer Science, Electrical and Space Engineering, Luleå University of Technology, Sweden
Email: udayanto.atmojo@aalto.fi; kashif.gulzar@aalto.fi; vyatkin@ieee.org

***Abstract*—This article presents the development of a distributed manufacturing case study enhanced with features that enable flexibility during the production process and the capability to continue the production process in case of fault scenarios. The approach described in this paper presents solutions to achieve the production of customized products, handle changes in product order, and minimize downtime and avoid total shutdown of the manufacturing system due to the occurrence of failures during the production process.**

Keywords—flexible manufacturing systems, IEC 61499; fault tolerance; reconfigurable manufacturing systems

I. INTRODUCTION

The focus of traditional manufacturing systems has always been towards achieving quantity (i.e. mass productions) of homogeneous products. Recently however, arises new manufacturing paradigm where modern manufacturing systems need to be capable of producing products with specifications provided by customers (customised products). This leads to new challenges and requirements that differ from traditional manufacturing systems which are typically 'static' and don't change during their operation. For example, when there are requests/orders to manufacture customised products, the manufacturing system should be 'flexible' (able to 'change') in order to achieve the required operations to manufacture products. Such changes in the manufacturing systems should occur 'on the fly' without restarting the system (or parts thereof) and incurring significant downtime, since downtime can or eventually lead to lost in revenues and valuable resources.

In addition, failures are also seen as a cause for downtime. Traditional manufacturing systems tend to be centralized, where a single controller (e.g. Programmable Logic Controller/PLC) is connected to all manufacturing/mechatronic devices and run software which govern and control the behaviours of manufacturing/mechatronic devices. Centralized architecture introduces single point of failure, which in this regard, the manufacturing system becomes prone to significant downtime when the sole controller experiences hardware/software failures. Traditional, centralized manufacturing systems typically are based on the 'legacy' approach of IEC 61131-3 [1], often in the form of PLC ladder programming logic.

On the other hand, the paradigm of modern manufacturing systems attempts to migrate from centralized architecture and implement decentralized, distributed manufacturing architecture where manufacturing/mechatronic devices are connected to more than one hardware controllers (e.g. PLCs) in a bid to negate single point of failure (and consequently, minimize downtime). As an example, with distributed setting, in case one of the hardware controllers fails, the other hardware controllers can take over and retain production process. This example shows how fault tolerance can be supported by having distributed architecture. However, the legacy IEC 61131-3 approach is not designed for distributed setting. Meanwhile, the newer IEC 61499 [2] software approach has been introduced, which is designed for modular, distributed setting. Hence, for modern manufacturing systems, the IEC 61499 approach is more suitable compared to the IEC 61131-3.

There have been works which investigate different methods to achieve flexibility and fault tolerance in distributed manufacturing systems. The IDEAS plug and produce project [3] presents a distributed manufacturing shop floor where the system is capable of self-organize using multi-agent control approach. Fault tolerance can be supported through reconfiguration of the factory floor on the fly when disturbances or unpredicted behaviours are detected. The CassaMobile project [4] used service oriented architecture (SOA)-based approach in the development of a flexible and modular production system that is environmentally friendly. A flexible production system utilizing web services approach is developed in the FLEXA project [5]. The IMC AESOP project [6] continues what the SOCRADES project [7] left off and utilizes different technologies (e.g. SOA, multi-agent, cloud) in realizing flexible and reconfigurable manufacturing system. The SelSus project [8] presents a web service-based approach for data acquisition techniques in distributed, self-healing production systems. The Self-Learning project [9] utilizes service-oriented based approach in integrating different supplementary processes in distributed manufacturing system and allows for dynamic reconfiguration of manufacturing/mechatronic machines. Some other works such as [10] use service oriented approach to support reconfiguration and flexibility in a IEC 61499-based distributed automation case study, while [11] showcases how to achieve flexibility, fault tolerance, and dynamic reconfiguration in a distributed manufacturing case study using the SOSJ framework [12], a programming framework achieved through the synergy of the SOA paradigm and a formal system-level language SystemJ [13].

This paper presents the development of a distributed assembly line case study enhanced with support for flexibility and fault tolerance. The case study extends the EnAS demonstrator [14] with the use of multiple distributed hardware controllers and is equipped with wireless communication capability. The paper is organized as follows. Section II briefly introduces the EnAS manufacturing demonstrator which realizes the assembly line manufacturing case study. Section III presents how flexibility in product manufacturing and the support for fault tolerance are introduced in the EnAS assembly line. Finally, concluding remarks are given in Section IV.

II. EnAS Demonstrator as Assembly Line Case Study – Brief Overview

The original physical layout of EnAS demonstrator is shown in Fig.1. Referring to Fig.1, the EnAS system is an assembly system comprising a conveyor system formed by six conveyors (each installed in Section 1, 2, 3, 4, 5, and 6), mechatronic stations consisting of two grippers (indicated by A and B) and two jack stations (indicated by C and D), and ten photocell sensors to detect the position of pallets on the conveyor system. Each jack station has a sledge, Sledge 1 and Sledge 2 which are used to put workpieces (which are taken from or to be put in the tin cans by the jack stations).

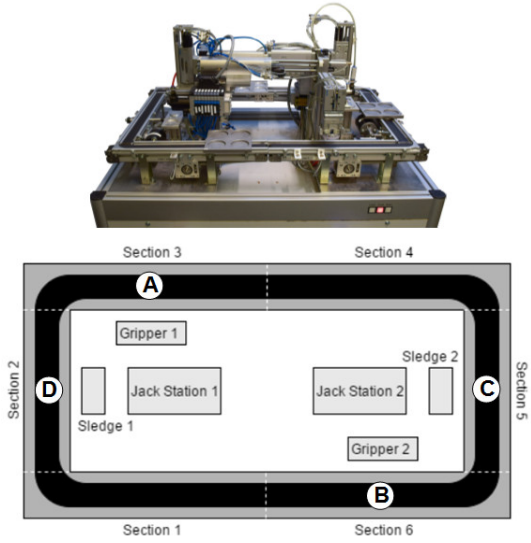

Fig. 1. The EnAS demonstrator from the side (top) and the top (bottom).

Workpieces can be put into the tin cans, where each tin can may be capped with a round cap. When on the conveyor system for transporting throughout the assembly line, tin cans and caps are put on pallets and are transported in a clockwise direction. The pallet's movement may be stopped when it reaches certain locations (e.g. when the pallet is adjacent to a mechatronic processing station) through selective switching of the conveyor sections. Each pallet has a total of four slots, forming two pairs of larger and smaller slots, respectively. The larger slots are used to hold caps while the smaller ones are reserved for tin cans, where each cap can be put on top of a can. Thus, two pairs of covers and tin cans can be transported by each pallet.

Improvements were done over the EnAS demonstrator in [15], where six IEC 61499-compatible NXT DCS mini controllers (of the nxtControl) have been deployed and run software behaviours that govern the individual mechatronic machines in the EnAS demonstrator. These controllers are interconnected through wired or wireless network. The improvements also include facilitate the possibility to update parts of the EnAS system on the fly to certain extent. The software (based on the IEC 61499 function blocks) implementation which governs the behaviour of the overall EnAS system in Fig.1 has been described in [15].

III. Introducing Flexibility and Fault Tolerance in EnAS Assembly Line

The previous attempt of improving the EnAS demonstrator in [15] added certain extent of reconfiguration support, however little attention was paid on handling product customization and changes of product specifications on the fly (flexibility). Also, the support for fault tolerance is limited. This work enhances the EnAS demonstrator with functionality which supports flexibility in production process and introduces certain degree of fault tolerance. With regard to this work, the support for flexibility covers the feature to alter product specifications during the production process and to detect faulty products and perform operations to re-assemble the products to match the actual specifications when possible. Meanwhile, the support for fault tolerance is introduced by utilizing distributed architecture where individual hardware controllers can take over in handling manufacturing operations in case one of the hardware controllers fails. To realize the aforementioned features, the EnAS assembly line is extended with some additional functions, resulting in a new assembly line with physical layout shown in Fig.2

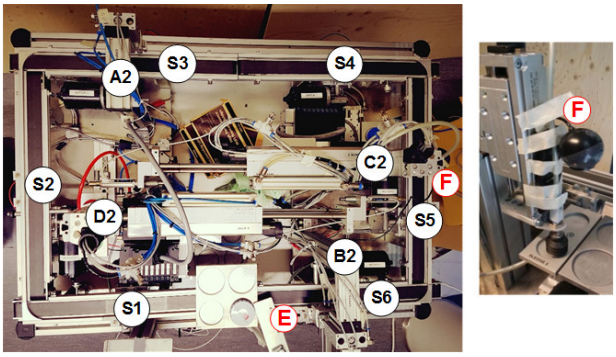

Fig. 2. Extended EnAS assembly line from the top (left) and camera (right).

The indicators A2, B2, C2, D2, S1, S2, S3, S4, S5, and S6 in Fig.2 (left) correspond to A, B, C, D, Section 1, 2, 3, 4, 5, and 6 in Fig.1 (bottom), respectively. As seen in Fig.2, the location of additional mechatronic devices is indicated in E and F (marked with red colour). A rotary arm is added in location E to take finished products from the pallet on the conveyor system for storage (e.g. then delivery to the customer), and a camera in location F (Fig.2 on the right) which performs vision-based sensing used to check whether the product contains the workpiece with the colour according to customer's specifications.

In this work, the EnAS assembly line utilizes a total of ten NXT DCS mini controllers. Four are assigned as 'master controllers' or denoted as Master Terminal Units (MTUs), while

six others are used 'slaves' for handling input and output (I/O) interfacing or denoted as Remote Terminal Units (RTUs). Six individual RTUs are responsible for providing I/O interfacing on Section 1, 2, 3, 4, 5, and 6, respectively.

All four MTUs run function blocks which handle higher-level logic that govern the behaviour of the entire EnAS assembly line system. They are able to access and manipulate physical I/O interfaces through the RTUs. In this work, each MTU is able to access the I/O data from all six RTUs. It is possible for one MTU to manipulate the output (I/O) value of all six RTUs. On the other hand, RTUs are physically connected to the sensors and actuators in the EnAS system, where outputs from the sensors are connected to the RTU physical input pins and inputs of the actuators are connected to the RTU physical output pins. Each RTU broadcasts its input values to all MTUs while it receives control signals (i.e. for actuation) from one MTU. MTUs performs synchronization (i.e. communicate with each other) so all MTUs will have the same state. During design time, individual RTUs are statically assigned to which MTUs they will be coupled (referred to as "default master" configuration).

A. Product Customizations during the Production Process

The support for product customizations is introduced by enabling the assembly line to produce two types of products, each with different coloured workpieces, i.e., blue and red workpieces. The final products will have the workpieces put inside tin cans and then covered by caps. Fig. 3 shows where tin cans and caps are put on the pallet and possible product variations in the EnAS assembly line.

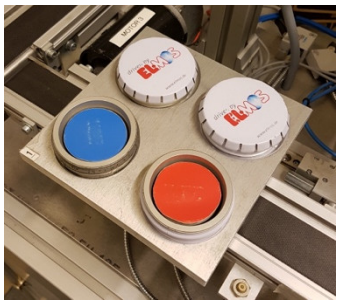

Fig. 3. Products on the pallet and their variations

Based on the customer's product specifications, the assembly line performs manufacturing operations to produce the product according to the specifications. Each product is assigned with a unique name (product ID) and is associated with the specifications given by the customer. A functionality, namely the product order handler, is implemented which monitors the production process of ordered products and issues commands to manufacturing functionalities in individual sections accordingly (i.e. based on provided product specifications and the progress of the production process of the corresponding product). Synchronization mechanism between MTUs and RTUs (which is detailed in Section III.C) allows the product order handler to track the progress of the production process of each product order.

As a scenario example of product customizations, an order has been requested by a customer to produce a product containing the red workpiece. The order is assigned with the name Order1. After the order is given to the assembly line and materials have been provided, the assembly line begins its operation. Referring to Fig.2, the pallet is moved by the conveyor system and stops at the point adjacent to the jack station in location D2 (when detected by the photocell sensor in location D2). Then, the jack station in D2 picks up the red workpiece which is put on one of the slot in Sledge 1 and then puts it on the tin can. Next, the jack station caps the tin can containing the coloured workpiece. Finally, the production process (which corresponds to the product order) is finished once the pallet reaches the rotary arm in E and then is taken from the pallet by the rotary arm.

In the context of the aforementioned scenario example, if during the production process the customer wishes to have a workpiece with a different colour (from red into blue) inside the tin can, the pallet with the tin can is routed towards jack station, e.g., in C2. The jack station in C2 will take the workpiece from the tin can. Assuming the tin can has been capped, the jack station removes the cap from the tin can before taking the workpiece from the tin can. Once the jack station removes and puts the previous workpiece temporarily on the Sledge 2, the jack station puts back the cap into the tin can. Then, the pallet is transported back towards the jack station in D2. When the pallet reaches the jack station in D2 and has the jack station uncap the tin can, the jack station takes a workpiece with blue colour from the other slot in Sledge 1 and puts it inside the tin can. The jack station then puts back the cap into the tin can, and then the conveyor can move the pallet towards the rotary arm for the product to be picked up from the pallet and then stored temporarily (e.g. before being delivered to the customer).

The scenario example of handling changes in product specification in the previous paragraph doesn't utilize the vision-based sensing provided by the installed camera, with the assumptions that the EnAS system has the information on where the coloured workpieces are put in which slots (predetermined slots) in the Sledge 1 and Sledge 2. If this information is not available to the EnAS system, the EnAS system is able to utilize the vision-based sensing function by using the camera to identify the workpiece colour inside the tin can on the pallet on the conveyor (colour detection). In this case, the product needs to be transported to the jack station in D2, which is adjacent to the camera. Then, the results from the colour detection function can be used, e.g., for the scenario in the previous paragraph and also for detecting incorrect workpiece (wrong colour) inside the tin can. More details regarding the colour detection function is provided in Section III.B.

B. Colour Detection Mechanism

In this work, a web camera is used as the vision sensor to gather image data/information. The camera is put in location F in the overall EnAS assembly system, attached to the jack station in C2, as seen in Fig.2. The camera is connected to a computing machine (PC, but it can also be a powerful embedded platform such as Raspberry Pi or Beaglebone Black) which executes a Python script that performs colour detection function. The Python script communicates with the IEC 61499 program via Ethernet using UDP/IP. Fig.4 shows the snapshot of the function blocks which trigger the colour detection function.

Referring to Fig.4, the function block FB1 is responsible for sending a command via UDP/IP to the colour detection function (i.e. Python script executed on PC) to capture an image using the web camera. FB5 determines whether within 5 seconds the image capture process has been successfully performed (after the command to capture an image is sent by FB1). Part of the IEC 61499 program which receives the result from the colour detection function and interprets it for the rest of the program in the EnAS system is shown in Fig.5. As shown in Fig.5, FB8 denotes the function block which receives results from the colour detection and buffers it to FB9. FB9 interprets and forwards the result to the other parts of the program (e.g. the product order handler).

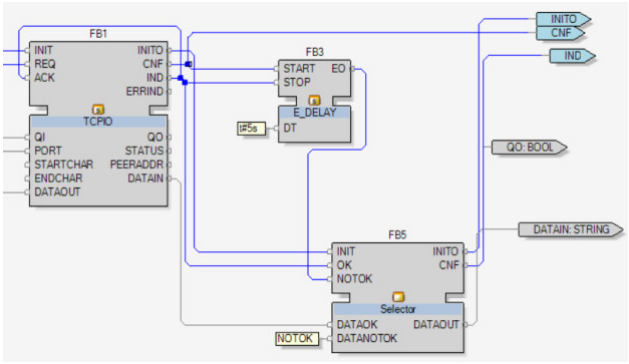

Fig. 4. Part of the IEC 61499 program that triggers colour detection function.

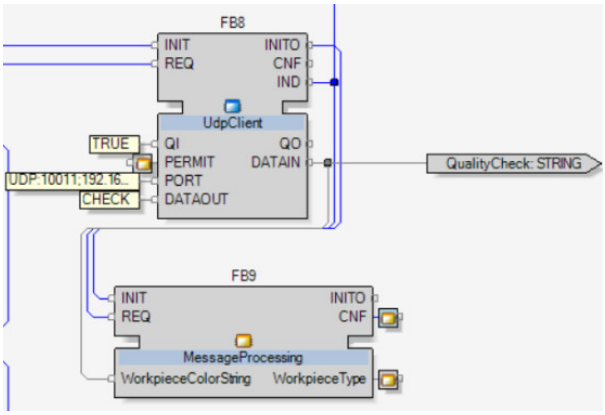

Fig. 5. Part of the IEC 61499 program that receives results from colour detection.

The colour detection function implemented in Python language complies with the flowchart shown in Fig.6. Once the execution starts, the Python script initializes the camera and UDP/IP server which will receive trigger. The script then waits to receive trigger. Upon receiving a trigger sent from the function block FB1 shown in Fig.4, the Python script sends a control signal to the camera to capture an image. Then, the image is converted into HSV (Hue, Saturation, and Value) format and then cropped to obtain the Region of Interest (ROI). Next, the image data undergoes filtering, histogram extraction, and thresholding processes, each for red and blue colour case (denoted as the 'Filtering, Extraction, Thresholding (Colour*)' process in Fig.5, where colour* denotes either red or blue, nevertheless the process is performed for both colours) to identify the colour of the workpiece inside the tin can. The result of the colour detection function is then transferred to the IEC 61499 program via UDP/IP.

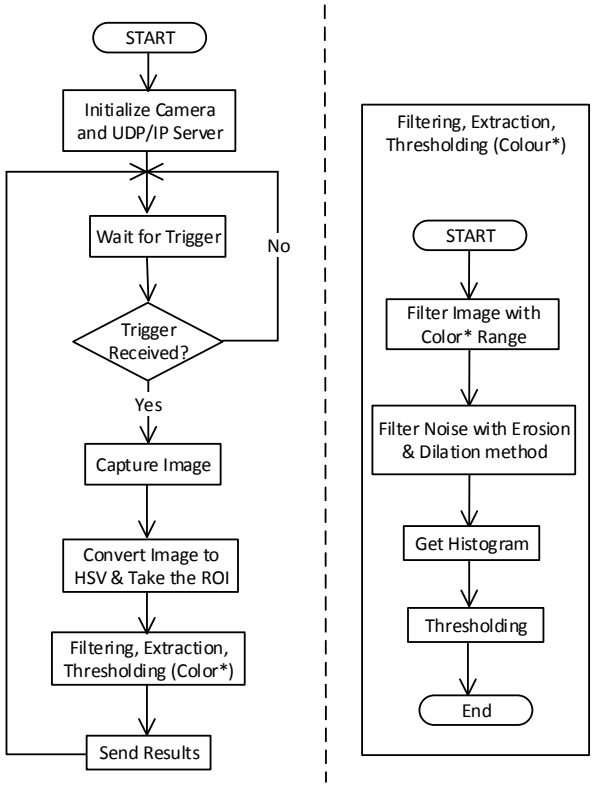

Fig. 6. Flowchart of the colour detection function.

C. *Support for Hardware Fault Tolerance*

In this work, the support for hardware fault tolerance is enabled through the interaction between MTUs and RTUs. How MTUs and RTUs are 'coupled' with each other is done through peer-to-peer publish-subscribe method, which includes heartbeat mechanism to establish loose coupling communication and to detect failures in MTUs (It is worth noting that this work assumes that failures only occur on MTUs. Any scenarios of failure which occur on other components or parts in the system is out of the scope of this work).

The responsibility in handling the interaction between MTU and RTU lies in the composite function blocks, namely the MTUPubSubCommunicationHandler and the RTUPubSub-CommunicationHandler, respectively. The MTUPubSub-CommunicationHandler function block runs on each MTU, while the RTUPubSubCommunicationHandler function block runs on each RTU, respectively. The snapshot of these function blocks is shown in Fig.7.

The interaction between MTU and RTU complies with the sequence diagram shown in Fig.8. In order to not overcomplicate the illustration, the sequence diagram shows and considers only the interaction between 2 MTUs, namely MTU1 and MTU2, and 2 RTUs, namely RTU1 and RTU2.

Referring to Fig.8, if a MTU is 'free' (not coupled to any RTUs), it will transmit a message to inform RTUs that it wants to connect to a RTU. Such message is denoted as 'Request Connect', which can be transmitted through broadcast or

unicast. Upon receiving 'Request Connect' message, a 'free' RTU (i.e. currently not coupled to any MTUs) will send a 'Response OK' to the MTU to inform the MTU that the RTU is not coupled to any MTUs and can form a master-slave relationship with the MTU. Upon receiving a 'Response OK' for the 'Request Connect' it sent beforehand, the MTU can proceed to establish the master-slave relationship with the RTU by transmitting a 'Connect' message and receiving a 'Response OK' from the RTU. A 'Request Connect' message can also be transmitted through unicast by a MTU to a specific RTU, which if the corresponding RTU is free, it will send a 'Response OK' to the MTU. With the successful transmission of 'Connect' and receiving of 'Response OK', the MTU and RTU is considered as 'coupled' as master and slave. It is possible for a MTU to have multiple slaves (i.e. RTUs) at a time.

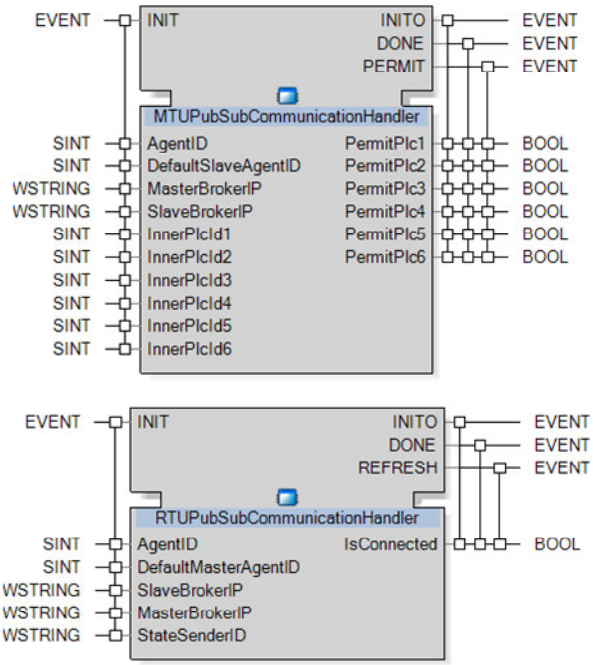

Fig. 7. The MTUPubSubCommunicationHandler (top) and RTUPubSub-CommunicationHandler (bottom) function blocks.

When a MTU is coupled to RTU(s), it periodically sends a heartbeat signal every (approximately) 500 milliseconds to its slave(s) to inform them that it is still 'alive'. If the RTU(s) don't receive heartbeat signal after (approximately) 1 second, the RTU(s) assume that the MTU fails and they will uncouple from the MTU. In Fig.7, this uncoupling process is represented by the 'Disconnect(MTU1)', which shows RTU1 attempting to uncouple from MTU1. Once the RTU(s) have been uncoupled from the MTU, they will broadcast a message to all MTUs that they are 'free' (not coupled to any MTUs). In Fig.7, this is represented by the RTU1 broadcasting 'Free' message after it has been uncoupled from MTU1. MTUs that receive this message. In this case study, given that there are MTU1, MTU2, MTU3, and MTU4 and if MTU1 fails, MTU2 will be given the responsibility to take over. Upon receiving 'Free' message, MTU2 sends a 'Request Connect' to RTU1 and then after receiving 'Response OK' from RTU 1, coupling between MTU2 and RTU1 is made through the successful transmission of 'Connect' and receiving of 'Response OK' associated to the 'Connect' message between MTU2 and RTU1.

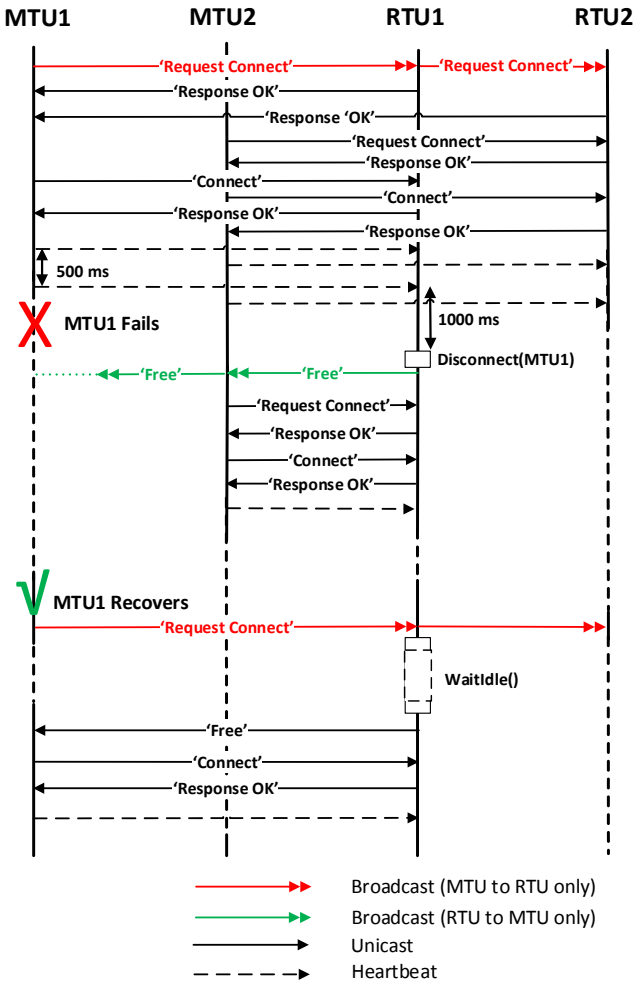

Fig. 8. MTU – RTU interaction sequence diagram.

During runtime, in case the failed MTU recovers and becomes operational, it will broadcast 'Request Connect' message. RTUs which have their 'default master' configuration set to these MTUs but currently are coupled to other MTUs will need to wait until the corresponding sections (i.e. Section 1, or 2, and so on until Section 6) they are responsible for don't process any product orders (represented as WaitIdle() in Fig.8). Once their assigned sections are not processing any product orders, the RTUs will transmit 'Free' message to the sending MTUs to inform that coupling procedure can be initiated. Once the MTUs receive the 'Free' message, they may establish master-slave relationship by sending 'Connect' message and then receiving 'Response OK' from the RTUs.

In addition, as part of the essential functionalities that provide fault tolerance in the EnAS assembly line, synchronization mechanism is enabled between MTUs for them to have the same knowledge regarding the state of all sections of the EnAS assembly line system. In this case study, synchronization involves the following: All inputs from the RTUs (which correspond to sensors in the EnAS assembly line

system) to all MTUs, interaction between the software behaviours that govern one section and the other software behaviours that govern the adjacent sections, and the product order handler functionality. Synchronization is essential in order for the software controllers running on individual MTUs have the same state. The synchronization between MTUs with regard to attributes required by the product handler functionality (which include information regarding current product order being produced, pallet locations, product specifications, and state of execution of the program on the MTU of each section) is also necessary when one MTU needs to take over the product order handling function from a faulty MTU. Note that this is possible since the product order handler functionality is present in each MTU, however at a time, the task of executing the product order handler functionality lies in one MTU.

IV. Concluding Remarks

With the increasing demand in requests for customised products from customers and the importance of minimizing downtime (e.g. due to failures), modern manufacturing systems need to differ from traditional manufacturing systems which typically produce homogeneous products in mass quantity and possess single point of failure due to typically having centralized architecture. This paper described the development of a distributed assembly line case study with is able to produce customised products, handle changes in the product order (e.g. changes in product specification), and has inherent support for fault tolerance. The production of customised products in the assembly line is supported by product order handler function which is able to monitor and track the progress of the production process due to synchronization mechanism and the interaction between MTUs and RTUs. With the addition of a camera, colour detection function can be added to the assembly line which introduces additional support in handling changes in product order. The interaction protocol between MTUs and RTUs and synchronization mechanism in the case study allow for certain degree of fault tolerance by allowing MTUs to take over the operation of faulty MTUs. Future works may include introducing mechanisms to deal with failures (of the MTUs) which occur before synchronization among MTUs is completed. Also, how the proposed approach satisfies 'hard' real time requirement can be investigated, however the result of such investigation would depend on and should (ideally) be supported by the infrastructure, e.g., the use of real-time-capable communication protocol and real-time-suitable execution platforms.

References

[1] K. H. John and M. Tiegelkamp, IEC 61131-3: Programming Industrial Automation Systems Concepts and Programming Languages, Requirements for Programming Systems, Decision-Making Aids: Springer Publishing Company, Incorporated, 2010.

[2] V. Vyatkin, "IEC 61499 as Enabler of Distributed and Intelligent Automation: State-of-the-Art Review," Industrial Informatics, IEEE Transactions on, vol. 7, pp. 768-781, 2011.

[3] M. Onori, A. Maffei, and F. Durand, "The IDEAS Plug & Produce System," in NewTech 2013 Advanced Manufacturing Engineering and Technologies, Stockholm, Sweden, 27-30 October, 2013, 2013, pp. 339-346.

[4] J. Friedrich, S. Scheifele, A. Verl, and A. Lechler, "Flexible and Modular Control and Manufacturing System," Procedia CIRP, vol. 33, pp. 115-120, 2015/01/01/ 2015.

[5] P. Webb and S. Asif, "Advanced flexible automation cell," Innovation for Sustainable Aviation in a Global Environment: Proceedings of the Sixth European Aeronautics Days, Madrid, 30 March-1 April, 2011, p. 296, 2012.

[6] S. Karnouskos, A. W. Colombo, T. Bangemann, K. Manninen, R. Camp, M. Tilly, et al., "The IMC-AESOP architecture for cloud-based industrial cyber-physical systems," in Industrial Cloud-Based Cyber-Physical Systems, ed: Springer, 2014, pp. 49-88.

[7] A. Cannata, M. Gerosa, and M. Taisch, "SOCRADES: A framework for developing intelligent systems in manufacturing," in Industrial Engineering and Engineering Management, 2008. IEEM 2008. IEEE International Conference on, 2008, pp. 1904-1908.

[8] M. S. Sayed, N. Lohse, N. Søndberg-Jeppesen, and A. L. Madsen, "SelSus: Towards a reference architecture for diagnostics and predictive maintenance using smart manufacturing devices," in 2015 IEEE 13th International Conference on Industrial Informatics (INDIN), 2015, pp. 1700-1705.

[9] D. Stokic, S. Scholze, and J. Barata, "Self-learning embedded services for integration of complex, flexible production systems," in IECON 2011 - 37th Annual Conference of the IEEE Industrial Electronics Society, 2011, pp. 415-420.

[10] W. W. Dai, V. Vyatkin, and J. H. Christensen, "The application of service-oriented architectures in distributed automation systems," in Robotics and Automation (ICRA), 2014 IEEE International Conference on, 2014, pp. 252-257.

[11] U. D. Atmojo, Z. Salcic, and K. I. K. Wang, "Dynamic online reconfiguration in manufacturing systems using SOSJ framework," in 2016 IEEE 14th International Conference on Industrial Informatics (INDIN), 2016, pp. 695-698.

[12] U. D. Atmojo, Z. Salcic, and K. I. K. Wang, "SOSJ: A new programming paradigm for adaptive distributed systems," in Industrial Electronics and Applications (ICIEA), 2015 IEEE 10th Conference on, 2015, pp. 978-983.

[13] A. Malik, Z. Salcic, P. S. Roop, and A. Girault, "SystemJ: A GALS language for system level design," Computer Languages, Systems, & Structures, vol. 36, pp. 317-344, 2010.

[14] (13 January 2018). Enas-energieautarke aktoren und sensoren. Available: https://www.energieautark.com

[15] L. Gröhn, S. Metsälä, M. Nyholm, L. Saikko, E. Väänänen, K. Gulzar, et al., "Manufacturing System Upgrade with Wireless and Distributed Automation," Procedia Manufacturing, vol. 11, pp. 1012-1018, 2017/01/01/ 2017.