Lyu, Tuojian; Blech, Jan Olaf; Vyatkin, Valeriy

A Case Study of Utilizing the SMT Solver for Deployment Optimization of an IEC 61499-based Application

Published in:
Proceedings of IEEE 30th International Symposium on Industrial Electronics, ISIE 2021

DOI:
10.1109/ISIE45552.2021.9576459

Published: 13/11/2021

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

Please cite the original version:
Lyu, T., Blech, J. O., & Vyatkin, V. (2021). A Case Study of Utilizing the SMT Solver for Deployment Optimization of an IEC 61499-based Application. In *Proceedings of IEEE 30th International Symposium on Industrial Electronics, ISIE 2021* (Proceedings of the IEEE International Symposium on Industrial Electronics). IEEE. https://doi.org/10.1109/ISIE45552.2021.9576459

A Case Study of Utilizing the SMT Solver for Deployment Optimization of an IEC 61499-based Application

Tuojian Lyu*, Jan Olaf Blech* †, 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: {tuojian.lyu}@aalto.fi, vyatkin@ieee.org

***Abstract*—The capabilities of dynamicity, flexibility and agile production are enablers to Smart Factory and Industry 4.0. One critical feature to achieve these capabilities is to achieve efficient deployment optimization for automation control applications. However, many factories have used control applications implemented using IEC 61131 rather than IEC 61499, even though IEC 61499 standard supports Smart Factory better than IEC 61131 standard with distribution and event-driven features.**

In this paper, we are studying a control application, named HotWaterTank, implemented via IEC 61499. This application can control a hot water tank via several buttons, such as inlet, outlet and heating buttons. Three steps are carried out to deployment optimization with this application that are 1) Flattening, 2) Penalty matrix calculation, and 3) SMT solver calculation.

Finally, from the experimental results, we prove that the deployment formulas we proposed can help achieve deployment optimization results while reducing the execution time of the SMT solver. Besides, we also discuss the parameter-related impact on the execution time and optimization results. Our contribution is to investigate the possibilities and difficulties of introducing the SMT solver and containerization technology to the IEC 61499-based application deployment optimization.

***Index Terms*—IEC 61499, satisfiability modulo theories, deployment optimization, flexible deployment, containerization**

I. INTRODUCTION

Smart Factory, as the core of Industry 4.0, introduces the need to adjust product type and manufacturing capacity in real time [1]. However, one noteworthy fact is that there are still many production systems developed for static environments that do not consider the increasing complexity and dynamicity of the production environment and extreme mass customization needs [2]. One example is that the centralized language, IEC 61131 [3], is still the dominant language for industrial automation application development, while the IEC 61499 [4] standard, with its distribution and event-driven features, is not widely used in industrial automation development [5]. Therefore, the traditional control systems based on IEC 61131 cannot sufficiently support agile production requirements, dynamic configuration, and deployment optimization for achieving Smart Factory. An essential capability in Smart Factory's vision is agile deployment optimization, and this requirement can be well supported by IEC 61499-based applications rather than IEC 61131 [6]. Therefore, the case used in our study is an IEC 61499 application that is to control a hot water tank.

Some difficulties are preventing current factories from achieving efficient and agile deployment systems as smart factories. One of the difficulties is the increasing number of software components (SWCs), and hardware components (HWCs) in manufacturing systems, which makes it challenging to find a widely applicable deployment mechanism and runtime optimization approach [7]. For large-scale control systems, implementing runtime deployment optimization is challenging because complex applications also have complex internal implementation. For example, the application used in our study has only five composite FBs, but the total number of basic FBs is 42. These internal FBs are called component FBs in IEC 61499 [8]. Since this paper is about the case study of the SMT solver applied in deployment optimization instead of introducing a complete flattening method, the execution model is the transparent container model, which means the event/data association and input sampling are not considered in our flattening process. According to the challenge of the internal complexity of current IEC 61499 applications, one of our goals is to reduce the execution time of optimization calculation.

The second difficulty is that the external environment tends to be more dynamic. Current control applications have been expected to perform dynamic processing and flexible configurations according to environment uncertainties [9] [10]. The highly dynamic environment makes deployment optimization more challenging. Therefore, deployment optimization is a long-standing research topic in Smart Factory and Industry 4.0.

Due to the advantages of the IEC 61499 standard, the application used in our study is implemented via IEC 61499. The IEC 61499 standard is gaining popularity as an enabler of Industry 4.0 due to its support for distribution, where control applications, presented as a network of communication software components, also known as FBs, can be easily deployed to different computing nodes, invoked in standard devices, and

† Deceased

978-1-7281-9023-5/21/$31.00 © 2021 IEEE

redeployed with a single click without affecting the application logic. Depending on various application complexities, the operations, such as adding, replacing or removing FBs, could be performed by means of the standard-defined management commands. Moreover, except for the management commands, there is no built-in feature of IEC 61499 to enable FBs migration, e.g., from one PLC1 to another PLC2 if PLC1 is down. This is also a motivation for us to propose the SMT solver to optimize the IEC 61499-based applications.

We will discuss the possibility of utilizing containerization technology as an enabler to improve deployment optimization calculation efficiency. Containerization technology will be applied in conjunction with SMT solver [11]. Specifically, by using the containerized IEC 61499 runtime, a large number of runtimes can be quickly extended to various devices with a single command. Containerization is now widely accepted as a powerful technology in software engineering due to its lightweight, flexibility, scalability, and short time-to-market. Although the technology originated in the IT world, it has recently gained some attention in embedded systems (e.g., [12]). Container technology can also potentially be used in industrial automation as an enabler for Smart Factory and Industry 4.0, as discussed by [13], [14], and [15]. In the context of IEC 61499, containerization can be applied to the device model and even to the resource model (resources are more fine-grained execution containers in IEC 61499, where a device can contain one or more independently executing resources).

In this paper, we address the difficulties (flattening composite FBs and FBs deployment optimization) as our scopes to test the proposed flattening method and SMT solver while reducing the execution time via introducing the containerization. The SMT solver is one such tool. To reduce the execution time, the runtime processes of IEC 61499 should be created and deleted dynamically, automatically and quickly.

The outline of this paper is as follows. Section II gives the details of a hot-water-control application developed based on IEC 61499 and describes the flattening pre-processing of the FBs. Section IV presents the mathematical model of the deployment optimization problem based on the SMT solver. Section V gives the experimental results based on the use case using our proposed SMT solver. Finally, section VI summarizes the performance of our work on the deployment problem based on the industrial IEC 61499 application and gives our plans for future work.

II. USE CASE: HOT WATER TANK

The control application is a model to control the inlet, outlet and temperature of a hot water tank. The model has six FBs, as shown in Fig. 1. These FBs control the water level and the water temperature via corresponding buttons. For example, the InFast button controls the fast inlet valve of the water tank; by turning this valve on, the water level in the tank will rise quickly. The FB, HotWaterTankModel, is a more complex FB that simulates the entire water tank, including an accurate reflection of the impact of fast/slow water inlet and outlet and

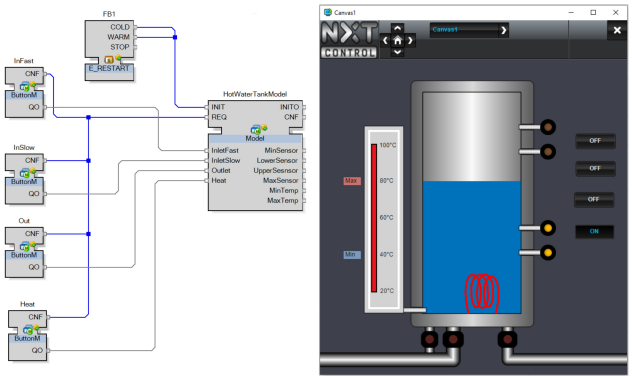

Fig. 1: The Hot Water Tank application in nxtStudio.

heater. With this model, the water tank operator can quickly observe water volume changes and temperature changes via Human Machine Interface (HMI) and control these changes in time to avoid hazards such as tank rupture and heater-broken.

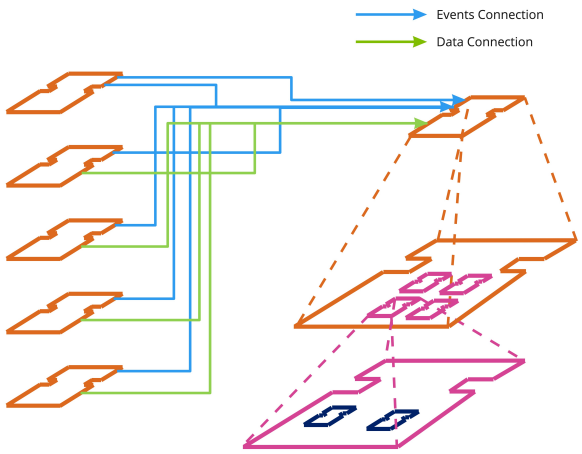

Fig. 2: The hierarchical viewpoint of the HotWaterTank application.

Fig. 1 shows the model in its most concise form, i.e., the internal complexity is abstracted through composite FBs. The composite FBs can also contain other composite FBs. Hence the whole model is implemented in a nested style. The hierarchical view of the model is shown in Fig. 2. Composite FBs is an essential type of FB in the IEC 61499 standard. Composite FBs provide a way to build more complex FBs from multiple basic FBs, finer composite FBs, or service interface FBs. Data connections and event connections for both input and output of composite FBs can be found within the corresponding connected component FBs. To optimize the deployment of internal basic FBs, this HotWaterTank application should be first flattened to obtain all the internal basic FBs and then perform the optimization to these flattened FBs rather than composite FBs.

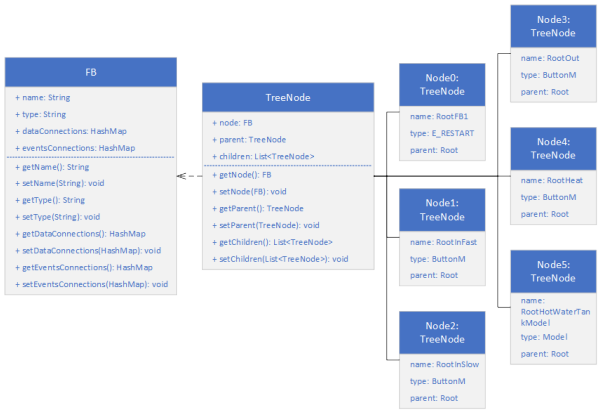

Fig. 3: The UML diagram of TreeNode and FB classes.

III. FUNCTION BLOCKS FLATTENING

The structure of many IEC 61499-based applications is similar to this HotWaterTank model, i.e., applications are implemented via a few simple and concise composite FBs. This design pattern enables developers to create more complex IEC 61499-based applications quickly because composite FBs only provide necessary interfaces to external FBs while abstracting the internal complexity.

However, when it comes to optimizing FBs deployment, these composite FBs make this requirement more challenging due to its abstraction. Because each composite FB has many component FBs inside, the internal configuration and implementation details should be provided from the application deployment perspective. Even though the use of many composite FBs simplifies the model and reduces the implementation complexity to a certain extent, it makes it more challenging to implement a deployment optimization mechanism for all FBs. For the deployment optimization of FBs, the granularity of FBs should be much finer to obtain a better performance.

Fine-grained optimization provides better resource allocation and management for the interests of basic FBs. Moreover, fine-grained deployment optimization allocates the resources of the devices more rationally to improve the performance of the entire application and system.

Therefore, an algorithm that flattens all the composite FBs is applied to obtain all the underlying FBs before proceeding with the deployment optimization using the SMT solver. We implement the flattening algorithm via a Java application. Specifically, we build a class FB in the java application, and this FB class in Java is the counterpart of FB in IEC 61499. On the other hand, a tree-node structure containing the FB class is introduced into the flattening application, i.e., each FB at a different hierarchical level has a node instance corresponding to it. The node instance contains an FB instance representing the FB with attributes such as FB name, FB type and connections. The UML diagram of the tree structure and the FB classes is shown in Fig. 3.

On the other hand, work [16] suggests using the data valves when flattening the hierarchical applications because the correct way to flatten the application should also consider the event-data associations. However, in this case, study, we only focus on the performance and results of the SMT solver and related deployment models/mathematical formulas instead of giving a complete flattening method. Therefore, in this paper, we will not go further on the topic of FBs flattening.

All the composite and flattened basic FBs of this HotWaterTank model are presented in Fig. 4. These blue FBs are the flattened basic FBs and do not contain the composite program blocks. When the whole HotWaterTank model is flattened with our flattening Java application, 42 basic FBs are essentially obtained to represent the whole system as the same as those composite FBs. These FBs will be used in the final deployment optimization algorithm. The deployment optimization result will suggest these 42 basic FBs for the destination of the device/container to where they should be deployed.

IV. DEPLOYMENT OPTIMIZATION MODEL

Referring to optimize the deployment, we propose a new deployment model (mathematical formulas) for three constraints and one objective. We argue that different deployment models/mathematical formulas for the same meaning of constraint or objective can significantly affect the execution time, optimization results and scalability. Deployment models that are not properly designed are less scalable, so they can only be used for simple control applications and are difficult to be applied to real complex control applications, e.g., the HotWaterTank application with 42 basic FBs. A well-designed deployment model can produce highly accurate results and improve the scalability and automation of the entire deployment process. Four deployment models are proposed to investigate four different deployment requirements. These four models are

- Constraint 1: A finite number of IEC 61499 runtime docker containers (variable n).
- Constraint 2: Maximum number of FBs per container (variable m).
- Constraint 3: Match the skills required for FBs (e.g., I/O capabilities) with the supported skills of the containers.
- Objective 1: Reduce the penalty scores by introducing the penalty matrix (matrix $Penalty$).

A. CONSTRAINT 1

For each FB, a variable x_i is created, and the value of each variable represents the container tag $(1, 2, ..., ContainerNum)$. For example, deploying FB x_i to container with tag "1" indicates $x_i = 1$. Finally, the deployment result calculated with the SMT solver is an assignment to all FBs x_i in a control application. Therefore, the assigned value for each FB should not be greater than $ContainerNum$. The deployment model for this constraint is shown in (1).

$$\forall x_i \in FunctionBlocks, \\ 1 \leq x_i \leq ContainerNum \tag{1}$$

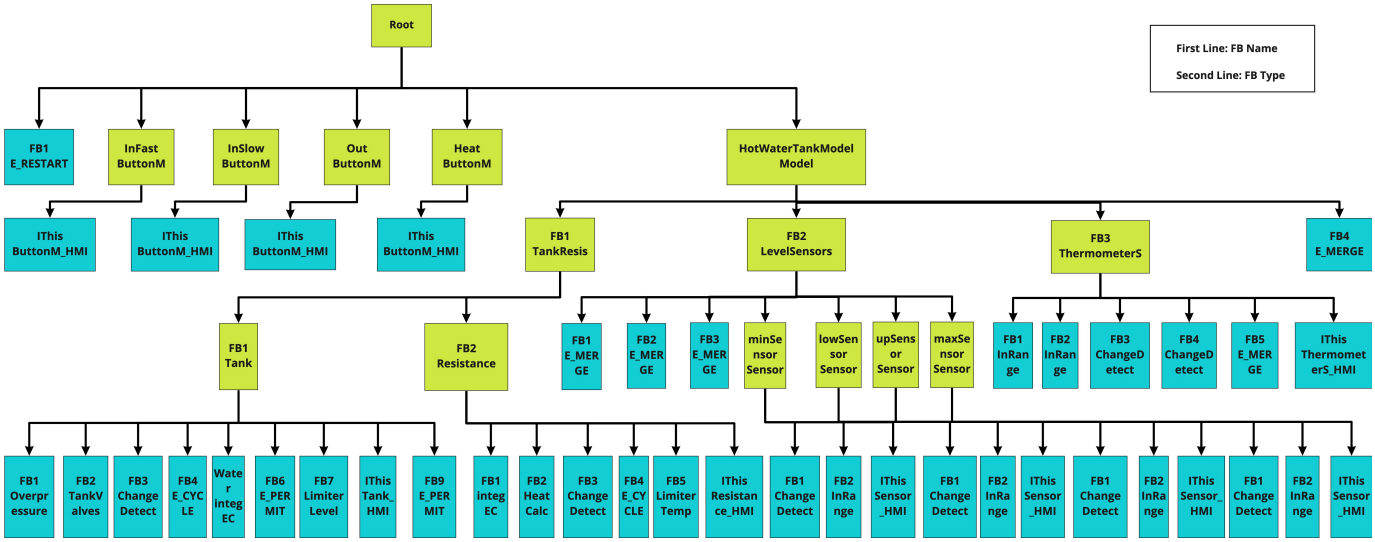

Fig. 4: The hierarchy of all FBs from the HotWaterTank model.

B. CONSTRAINT 2

The second constraint is that there is an upper limit to the number of FBs that is the maximum number of FBs each container can hold. The deployment model for this limit is shown in (2). For example, if the upper limit is two, then any three control application variables should not have the same value assigned. If three variables are equal, three FBs are running in the same container, i.e., the upper limit is exceeded.

$$\begin{aligned} &MaxFBs * ContainerNum \geq TotalFBs, \\ &\forall x_{i_0}, x_{i_1} \ldots x_{iMaxFBs} \in FunctionBlocks \Rightarrow \\ &(x_{i0} = x_{i1}) \wedge \ldots \wedge (x_{i0} = x_{iMaxFBs}) \neq true \end{aligned} \tag{2}$$

C. CONSTRAINT 3

The third constraint (3) indicates that the skill required for each FB should match the skill supported by the assigned container. For example, a FB that requires temperature sensor support needs to be deployed to a container with a temperature sensor via proper I/O ports. Otherwise, this FB can not read or write temperature data using the container's I/O successfully. The introduction of this formula further enhances the rationale for the entire deployment. The supporting skills should be considered before deploying FBs to destinations. The deployment results can be used to deploy FBs to containers while ensuring them working well as expected.

$$\begin{aligned} &\forall x_i \in FunctionBlocks \\ &Containers(1,2,3)\ support\ I/O\ skill, \\ &(swSkill\,(x_i) > 0) \Rightarrow \\ &(((x_i = 1)\ \&\&\ (swSkill\,(x_i) = hwSkill\,(1)) = true)\ || \\ &((x_i = 2)\ \&\&\ (swSkill\,(x_i) = hwSkill\,(2)) = true)\ || \\ &((x_i = 3)\ \&\&\ (swSkill\,(x_i) = hwSkill\,(3)) = true)) \end{aligned} \tag{3}$$

D. OBJECTIVE 1

The fourth formula (4) presents the optimization objective rather than the constraints. Once the number of FBs, the number of available containers, and the limit for each container are known, a new optimization objective is proposed. This optimization is how to allocate the FBs to minimize the total penalty scores based on all the above constraints.

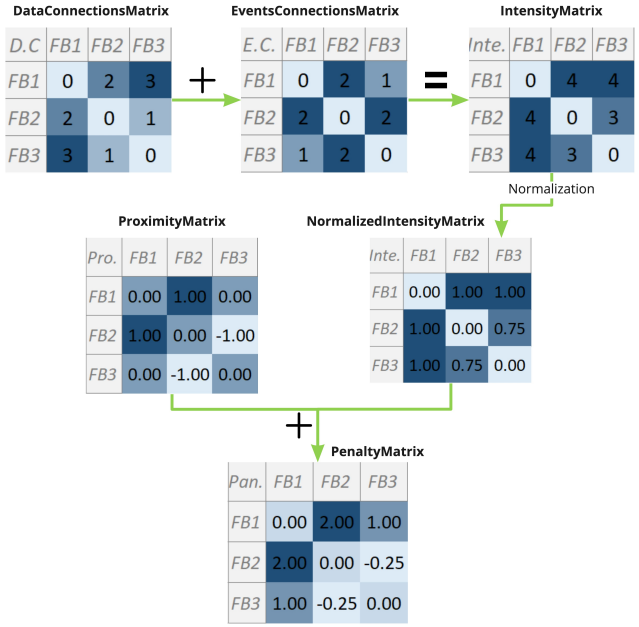

Fig. 5: An example of the process to obtain the penalty matrix.

This penalty matrix we proposed indicates how much important it that two FBs should be deployed to the same container/device. The $Penalty$ matrix is achieved by calculating the data and events intensity with the proximity requirements of all basic FBs. An example of the method to obtain the $Penalty$ matrix is shown in Fig. 5. The process to calculate the final penalty matrix is that the data connections and events connections will be added together to get the communication

intensity matrix. After that, the normalized intensity matrix can be obtained from the intensity matrix. The last step is to add the proximity matrix to the normalized intensity matrix so that the penalty matrix can finally be calculated.

There are many advantages of utilizing the penalty matrix instead of using data/event intensity directly. One of the advantages is that utilizing the data/event intensity matrix is shallow because more data/event connections do not mean more intensive or frequent communications between FBs. Suppose two FBs that need to communicate are assigned to two separate containers. In that case, they need to be externally communicated via additional service interface FBs. Our solution is that a weighted intensity matrix is used to represent external communication stress. This weighted intensity matrix refers to the number of data connections and event connections between each pair of FBs with different weights. Another advantage is that, depends on different cases, some FBs may be better deployed based on customized needs. For example, the Outlet button FB should be deployed with the Outlet valve FB as close as possible to reduce events transmission latency.

$$\forall x_i, x_j \in FunctionBlocks,$$
$$\min \sum_{x_i \neq x_j} Penalty\,[i]\,[j] \tag{4}$$

V. EVALUATION

To optimize the deployment of all basic FBs of this HotWaterTankModel application, the model should first be flattened. The flattening process will help to get all basic FBs. Redeploying a model means optimizing the deployment of all basic FBs instead of the composite FBs. Therefore, flattening all the composite FBs is the first step in the deployment optimization calculation before using the STM solver. The second step is to obtain a penalty matrix based on all the flattened FBs by calculating the weighted data connections, event connections and proximity matrix. Finally, based on this penalty matrix and the SMT solver application, the deployment optimization results are obtained.

A. FLATTENING RESULTS

Fig. 6 shows the results obtained after flattening the composite FBs of the HotWaterTank model by using a Java application. Each tree node contains a FB class representing the name, type, data connections and event connections of the corresponding FB. For example, the name of the FB of this node (28th tree node) is ”RootHotWaterTankModelFB1FB2FB5”, and the type is ”LimiterTemp.” The naming rule in this flattening application follows the absolute path of this FB. For example, the flattened FB with the name ”RootHotWaterTankModelFB1FB2FB5” indicates a FB (FB5) inside the composite FBs that are FB2, FB1 and HotWaterTankModel. Besides, it is known that this FB has three data connections and two event connections. Using HashMap, it is possible to match the source and destination for both event and data connections. Since there are 42 basic FBs, we will not list and analyze each final FB one by one. However, by analyzing

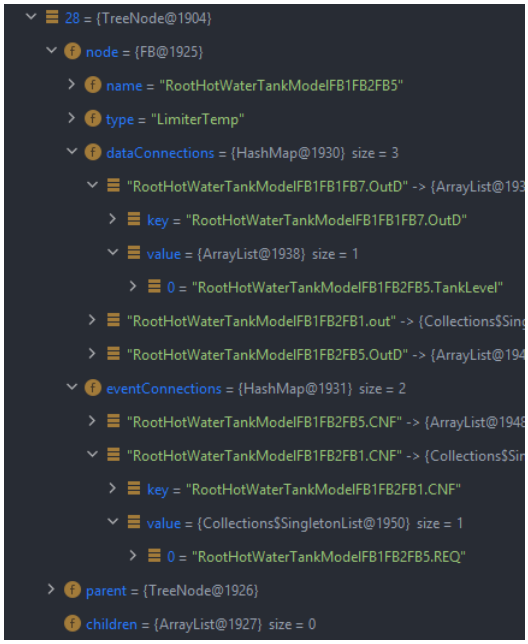

Fig. 6: The FBs-flattening results by using the Java application.

them against the model in nxtStudio, it is proved that the flattening algorithm and the Java application we implemented are correct to pre-process all the composite FBs in the model.

B. PENALTY MATRIX

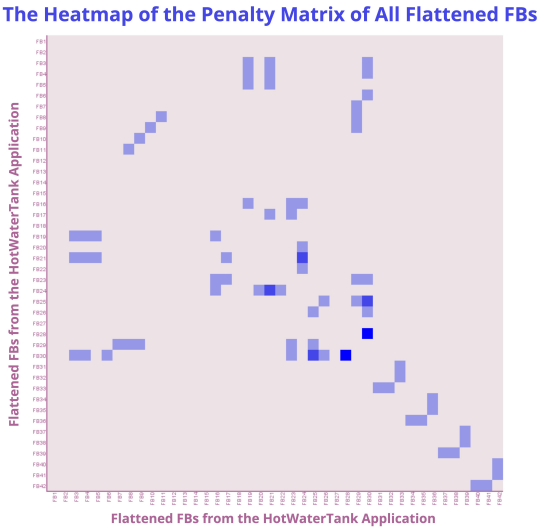

Fig. 7: The heat map of the penalty matrix obtained from the HotWaterTank model.

Once all the non-composite blocks are obtained, the penalty matrix can be obtained by calculating from the data connections, the event connections and the proximity matrix. Fig. 7 displays the penalty matrix of this HotWaterTank model using a heat map. The deep color block indicates that the two corresponding FBs should be deployed preferentially in the same container/device. As can be seen from this penalty matrix,

FB30 (RootHotWaterTankModelFB1FB2FB2) has communication with seven other FBs, so there are seven colored blocks on the heat map. Since it has three data connections and one event connection with the FB28 (RootHotWaterTankModelFB1FB2IThis), the total number of connections is the biggest among other connected FBs, and the weight is 1, so the color of this block is the one in dark blue.

C. DEPLOYMENT OPTIMIZATION RESULTS

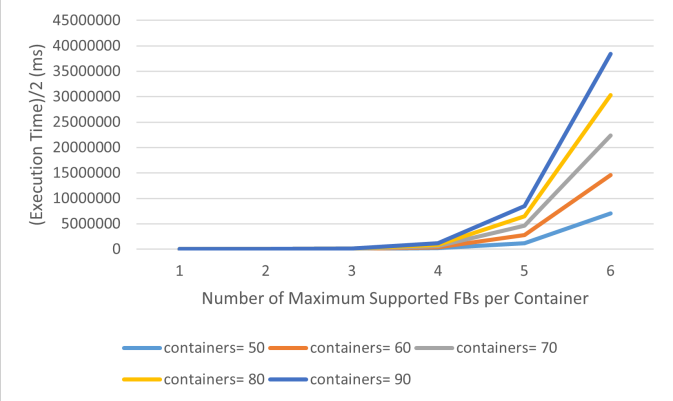

Fig. 8: The execution time of SMT solver according to different parameters.

Different parameter settings for the SMT solver and the corresponding mathematical formulas significantly affect the experimental results and execution time. Fig. 8 shows a significant difference in the execution time of the deployment optimization obtained for various parameters, such as the number of containers and the maximum FBs per container can support. For example, for the same number of containers of 90, the execution time is 45297ms and 8055853ms when the container capacity is three and six, respectively, because a better combination of parameters can significantly increase the execution efficiency of the SMT solver. For complex industrial control applications, efficient deployment optimization can better meet the requirements of the Smart Factory and Industry 4.0 for agile manufacturing. These experiments help us find the optimal parameter combinations for our SMT solver and deployment mathematical formulas in future work.

VI. CONCLUSION AND FUTURE WORK

As production environments become more complex and dynamic, Industry 4.0 demands more advanced agile production capabilities for achieving Smart Factory. To help traditional manufacturing factories upgrade to smart factories, improving the deployment performance is a critical step and capability. If an industrial control system cannot support distributed and agile production requirements, the upgrade to Smart Factory will be challenging. For this purpose, we analyze the deployment optimization of a control application (HotWaterTank) developed using IEC 61499 by using the SMT solver and the designed deployment optimization formulas.

We designed and implemented the flattening algorithm that flattens all-composite FBs from this application. The flattening application performs well in experiments, i.e., it provides accurate and detailed information to the SMT solver. Moreover, we also give mathematical formulas for the SMT solver and design, where different parameter combinations lead to different calculation results and execution time. Based on our experimental results, a more suitable combination of parameters can be obtained to research related problems.

We plan to improve further this SMT solver-based deployment optimization model for future work while introducing more deployment syntax. The attempt is to improve the flexibility with customized needs of the optimization while further reducing the execution time. Introducing more syntax for FBs deployment and designing the mathematical models are the main focus of future research.

REFERENCES

[1] H. Chen. Theoretical foundations for cyber-physical systems: a literature review. *Journal of Industrial Integration and Management*, 2(03):1750013, 2017.

[2] Z. Shi, Y. Xie, W. Xue, Y. Chen, L. Fu, and X. Xu. Smart factory in industry 4.0. *Systems Research and Behavioral Science*, 37(4):607–617, 2020.

[3] M. Tiegelkamp and K. John. *IEC 61131-3: Programming industrial automation systems*. Springer, 2010.

[4] V. Vyatkin. The iec 61499 standard and its semantics. *IEEE Industrial Electronics Magazine*, 3(4):40–48, 2009.

[5] U. Atmojo and V. Vyatkin. Programming future industrial cyber physical systems: A review and perspective on the state of the art. 2018.

[6] B. Chen, J. Wan, L. Shu, P. Li, M. Mukherjee, and B. Yin. Smart factory of industry 4.0: Key technologies, application case, and challenges. *IEEE Access*, 6:6505–6519, 2017.

[7] T. Terzimehić. Optimization and reconfiguration of iec 61499-based software architectures. In *Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, pages 180–185, 2018.

[8] A. Zoitl and L. Robert. Modelling control systems using iec 61499.

[9] Z. Zhang, X. Wang, X. Wang, F. Cui, and H. Cheng. A simulation-based approach for plant layout design and production planning. *Journal of Ambient Intelligence and Humanized Computing*, 10(3):1217–1230, 2019.

[10] H. Fatorachian and H. Kazemi. A critical investigation of industry 4.0 in manufacturing: theoretical operationalisation framework. *Production Planning & Control*, 29(8):633–644, 2018.

[11] C. Barrett and C. Tinelli. Satisfiability modulo theories. In *Handbook of Model Checking*, pages 305–343. Springer, 2018.

[12] H. Manninen, V. Jääskeläinen, and J. Blech. Performance evaluation of containerization platforms for control and monitoring devices. In *25th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2020, Vienna, Austria, September 8-11, 2020*, pages 1061–1064. IEEE, 2020.

[13] S. Kugele, D. Hettler, and J. Peter. Data-centric communication and containerization for future automotive software architectures. In *2018 IEEE International Conference on Software Architecture (ICSA)*, pages 65–6509. IEEE, 2018.

[14] S. Sarkar, G. Vashi, and P. Abdulla. Towards transforming an industrial automation system from monolithic to microservices. In *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 1256–1259. IEEE, 2018.

[15] N. Nikolakis, R. Senington, K. Sipsas, A. Syberfeldt, and S. Makris. On a containerized approach for the dynamic planning and control of a cyber-physical production system. *Robotics and Computer-Integrated Manufacturing*, 64, 2020.

[16] V. Dubinin and V. Vyatkin. On definition of a formal model for iec 61499 function blocks. *EURASIP Journal on Embedded Systems*, 2008:1–10, 2007.