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

SMT-based deployment calculation for IEC 61499 control applications

Published in:
Proceedings of 4th IEEE International Conference on Industrial Cyber-Physical Systems, ICPS 2021

DOI:
10.1109/ICPS49255.2021.9468194

Published: 05/07/2021

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). SMT-based deployment calculation for IEC 61499 control applications. In *Proceedings of 4th IEEE International Conference on Industrial Cyber-Physical Systems, ICPS 2021* (pp. 172-178). Article 9468194 IEEE. https://doi.org/10.1109/ICPS49255.2021.9468194

SMT-Based Deployment Calculation for IEC 61499 Control Applications

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 dynamic and flexible deployment optimization of industrial control applications is essential for achieving the goals of Smart Factories and Industry 4.0. In this paper, we are studying distributed IEC 61499 control applications. To achieve the required efficiency and flexibility of the control application deployment, we propose using a Satisfiability Modulo Theories (SMT) solver to solve the underlying redeployment model constraints. In our work the Z3 solver is used for this purpose. The proposed redeployment models represent the redeployment problem and allow the Z3 solver to efficiently and quickly calculate the optimization result. Furthermore, based on the Z3 solver APIs, several critical components of the redeployment architecture (Monitor, Parser, Generator, etc.) are implemented in Java applications. The experiment results indicate that, based on our implemented Z3-based software and redeployment models, the deployment optimization results can be quickly obtained for simple IEC 61499 applications.**

***Index Terms*—IEC 61499, containerization, flexible redeployment, smart factory, satisfiability modulo theories**

I. INTRODUCTION

Traditional manufacturing systems are designed and developed for static production processes and did not consider the dynamics [1] of their environment. As a result, the functionality of traditional systems cannot meet the requirements of trends such as the Smart Factory and Industry 4.0. These initiatives require efficiency and flexibility to reconfigure and optimize industrial systems [2]. As more consumers demand rapid and personalized products and services, the demand for these two critical features in industrial automation systems increases significantly. However, there are several difficulties in achieving efficiency and flexibility for current 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 redeployment architecture and runtime optimization methodology [3]. For more complex control systems and applications, runtime deployment optimization is often more challenging to achieve. Failure to calculate optimized redeployment dynamically and quickly can significantly degrade the performance of the Smart Factory. In addition to the increasing complexity of control applications, the external environment becomes increasingly complex and dynamic. Current control applications and systems are now required to have the ability to be dynamically processed and flexibly configured according to the environment dynamics [4]. The increasing complexity of applications and the highly dynamic environment make the optimization of redeployment increasingly difficult. This is why runtime redeployment has been a long-standing research issue for the Smart Factory in Industry 4.0.

The IEC 61499 standard [5], [6] is gaining popularity as an enabler of Industry 4.0 due to its support of distributed automation architectures, in which control applications, presented as networks of communicating software components (a.k.a. function blocks (FB)) that can be easily deployed to various computational nodes, called in the standard devices, and redeployed in one click without affecting the application logic. However, different mappings of function blocks to devices can clearly affect the overall system performance. Therefore, optimization of the FB deployment becomes an important problem in the IEC 61499 development context. Finding an optimal deployment is challenging due to the increasing number of function blocks in control applications. To find an optimization solution, the optimization problem needs to be formulated mathematically, better in the form of a known mathematical problem, for solving which a software tool (solver) would exist. The SMT [7] solver is one such tool.

Containerization is now widely accepted as a powerful technology in software engineering due to its features, such as lightweight nature, flexibility, scalability, and shorter time-to-market. While originating in the IT world, the technology has recently gained some interest in embedded systems (e.g., [8]). Container technologies can also be potentially applied in the industrial automation domain as an enabler of the Smart Factory and Industry 4.0 discussed by [9], [10] and [11]. For example, containerization can be used to create flexible control systems, scale up current control applications, simulate controllers' legacy engines, and shorten the time between development and (re)deployment. In the IEC 61499 context, containerization can be applied to the device model, or even to the resource model (resource is a finer-grain execution container in IEC 61499, a device can contain one or several independently executed resources). Containerization of IEC 61499 devices (resources) could provide the possibility to dynamically scale up runtime containers if the redeployment result is not satisfied.

† Deceased

In this paper, we investigate the problem of optimal deployment of IEC 61499 function block applications to the compliant devices. The method can be applied to both the traditional and containerized implementation of devices. Further in the paper, we refer to such devices as containers and assume them to be capable of hosting the IEC 61499 runtime (RT) environment, receiving FBs, and managing computational resources (CPUs, memory, and network) for FBs. Moreover, we propose new redeployment models of constraints and objectives for the redeployment problems. We also implement a supporting tool framework consisting of four components: Monitor, Coordinator, Parser, and Generator, implemented as Java applications. The Z3 [12] solver is used in our experiment as an efficient SMT solver tool. The role of the Generator is to translate redeployment models of the redeployment problems into formulas that the Z3 solver can understand and process. The Z3 solver then automatically calculates these SMT formulas based on proposed redeployment models. 4diac [13] is used as the IEC 61499 IDE for the control applications development due to its supported reconfiguration and redeployment mechanism.

The contributions of our research are the following four aspects: 1) a tool framework is implemented, consisting of Monitor, Coordinator, Parser, and Generator, to automate the translation of the deployment problem to the Z3 solver, 2) new redeployment models of the redeployment problems are proposed and analyzed, 3) applying Forte containers as the distributed 4diac runtime environment to receive FBs, 4) for control applications with various complexity levels, we present the redeployment calculation time based on the same architecture and the redeployment models.

The outline of this paper is as follows. Section II presents the related works regarding the industrial optimization and redeployment with Design Space Exploration (DSE), SMT, and agent-based technologies. Section III shows the architecture of the implemented Z3-based redeployment software and explains the functionalities of several components. Section IV provides details about the redeployment models we proposed to solve the redeployment problem with constraints and objectives. Section V presents the measuring results for our redeployment models and implemented tool framework and compares the latency results between two scenarios. Finally, VI concludes the performance of our work on the industrial IEC 61499-based redeployment problem and gives our plan for future work.

II. RELATED WORK

The optimization deployment problem is essentially a part of the architecture optimization problem [14]. The software optimization problem in the automotive domain has been widely discussed and studied. Therefore, some software optimization approaches in the automotive domain can be applied to the architecture optimization of industrial control applications. Some studies [15], [16], [17] discuss software optimization problems in the automotive domain using a combination of DSE and Model-based Design (MbD) with an SMT solver or linear programming to solve the related optimization problems. However, it is necessary to present research problems and research methods related to architectural optimization problems in the industrial domain. We study control applications based on the IEC 61499 standard for industrial control applications because this standard can sufficiently support the need for dynamic redeployment at runtime. Therefore, we discuss and study the optimization problem based on DSE and IEC 61499 dedicated industrial control applications for deployment.

Previous studies [18], [19], [20], [21] and [22] proposed the use of agent-based technologies in industrial control systems and applications. In particular, [23] introduced the idea of dynamic redeployment of automation agents encapsulated in IEC 61499 function blocks, which strongly motivates our current research. We use the framework of the general multi-agent system in our Z3-based software development, i.e., the components (Monitor, Parser, Generator, etc.) of the implemented software follow the multi-agent system architecture.

In [24], Goldschmidt et al. present an abstract architecture based on container virtualization technology. Based on container concepts, their proposed architecture addresses two concerns that are legacy emulation for controllers and flexible function deployment. Their work offers the possibility of applying container virtualization technology in the industrial automation domain. Due to containerization's efficient orchestration capability, containers can be dynamically started, scaled, redeployed, and deleted by simple commands provided by containerization tools (Docker/Kubernetes). On the other hand, manually configuring a significant number of devices with necessary RT, libraries and dependencies requires intensive engineering work and degrades the flexibility and efficiency of control systems.

Sinha et al. [25] propose a method to calculate optimal configurations that satisfy evolving control system requirements. An SMT solver is used to calculate the deployment based on SMT constraints in these evolving control system requirements. They conducted experiments on their proposed method based on a real-world airport baggage handling system. The performance of the incremental and compositional methods was compared based on the experiments, and better performance was achieved by applying the incremental method to the case of slight changes in the system. To improve the reliability of the distributed control application, a redundancy strategy was used, i.e., one to five copies of each function block. Thus, although only the affected components configurations need to be recalculated, there is a potential problem that all FBs may need to be redeployed due to additional constraints or objectives. For example, usually, optimization objectives are conflicting with each other, i.e., the order or weights of multiple objectives should be defined to avoid conflicts. Therefore, the newly introduced objectives may lead to potential problems with the current configuration of distributed FBs, i.e., all FBs may need to be redeployed due to different objectives.

Terzimehic et al. [26] present the DSE application to find

the optimal deployment of IEC 61499-based control applications. Besides, they show the applicability of the constraints, objectives, and annotations to the deployment problem. In experiments, they combine the DSE method with the IEC 61499 model and conclude that the method is well suited for Industry 4.0 and can be used for fault recovery without downtime. However, there is no concrete method proposed to achieve the optimization calculation based on the DSE automatically.

In their later study [27], based on the proposed framework and DSE, two objectives (functional coupling and end-to-end latency) are presented and created using redeployment models. Moreover, the Z3 solver results can prove the validity of these redeployment models. At the same time, they show that the SMT solver can be well applied to redeployment calculation based on IEC 61499 control applications. However, the redeployment process is still semi-automatic and does not discuss the possibility of using the Z3 solver to solve other objectives and constraints with more complex redeployment models. Also, the scalability of this approach is not investigated. For simple scenarios with only a few FBs, good results can be obtained in the appropriate time. However, more complex scenarios with dozens of FBs lead to unacceptable long processing latency. This long latency is not suitable for dynamic auto-redeployment scenarios that require runtime capability. Also, the workflow of their proposed architecture is still semi-automatic. The specific implementation of the SMT solver and the automatic processing of 4diac's .sys file is not explicitly given.

Our research focuses on further automating the entire redeployment calculation process and proposing new rational redeployment models to represent the redeployment problem. Specifically, firstly, two new components are introduced, namely, Coordinator and Parser, which can increase the efficiency, flexibility, and automation of the framework. Secondly, we propose new redeployment models. These redeployment models can be well applied to the Z3 solver to calculate the optimal deployment solution that is more reasonable and fast. It is also shown that the proposed redeployment models are highly scalable to other more complex redeployment problems.

III. PROPOSED FRAMEWORK

The architecture proposed in our work follows ideas from the automatic software systems [28]. In IBM's proposed architecture, the self-management capability of a control system enables the system's functionality to change in response to perceived changes in the environment. The automation capability is based on a control loop that collects information about the system and reacts accordingly. Based on the architecture proposed by IBM, we implemented the Java application as the Z3-based calculation software and its architecture is shown in Fig. 1. The Java application of Z3-based calculation incorporates container virtualization technology to deploy IEC 61499-based FBs to distributed containers. Since this paper focuses on proposing redeployment models for redeployment calculation and calculating these models in the implemented redeployment software, only Coordinator, Generator and Parser are introduced in detail.

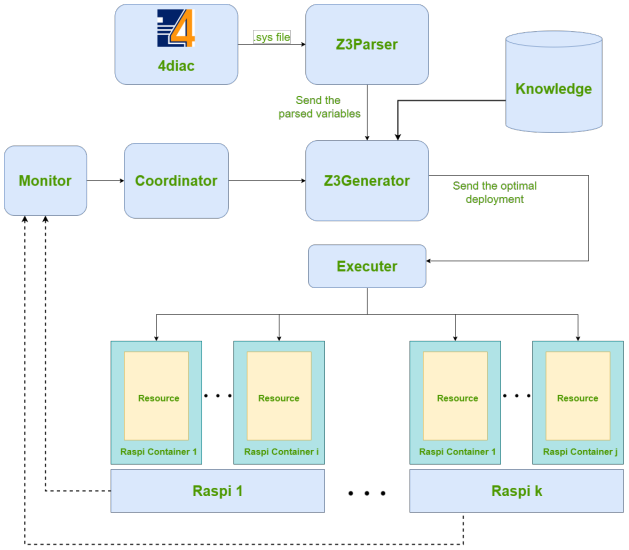

Fig. 1. The architecture of the redeployment software based on the Z3 solver.

A. Coordinator

The Coordinator receives and processes monitoring messages from the Monitor and provides feedback based on the monitoring information results, e.g., to initiate a redeployment request if the CPU or memory usage of a container exceeds a limit. There are two types of messages received in the Coordinator: the status of underlying devices (PLCs) and the containers running on the device. These two types of status information provide a bigger picture of the current control system and control application running on the device and the container. If an abnormal condition occurs in a container or device, the Coordinator can react quickly and appropriately based on the current conditions.

B. Generator

The Generator receives the redeployment request from the Coordinator and initiates a redeployment calculation process. A redeployment consists of three steps: Extracting the required redeployment models for the current devices and containers. Converting the adopted redeployment models into a format that can be processed by the Z3 solver. Invoking the Z3 solver to calculate the optimal redeployment based on the initialized conditions and selected objectives. Extracting the redeployment models is the most important of these three steps. This step requires multiple mathematical variables to represent each function block, and each redeployment model needs to be applied and constructed for each function block. The information for creating the redeployment models comes from Monitor's metrics and the initialized constraints of the current control application returned by Parser. Different redeployment models need to be constructed for various constraints and objectives, such as the constraint not having more than

five FBs per container or the objective minimizing the delay from the start to the end of an event. Until all reconfigurations are correctly represented in the redeployment models, the Z3 solver can understand and find the optimal deployment solution required.

C. Parser

The Parser is used to process the .sys file for the IEC 61499-based control application from 4diac. The .sys file is an XML file format that exhibits the current deployment of the control application, such as the number of FBs, the number of PLCs, and the data connections.

The metadata, e.g., the number of FBs, devices and data connections, is required by the Generator to initiate a redeployment calculation. For obtaining such metadata, the Parser is used to process the XML file (.sys file) and automatically retrieve and pass the metadata to the Generator. It should be noticed that the Parser will only be called when there is a change to the configuration of the control application, i.e., if there is no modification to the application, there will not be any metadata sent to the Generator. In other words, the Parser and Generator will remain idle if there is no redeployment request for the control application. This request-based parsing mechanism greatly increases the efficiency of the redeployment calculation.

IV. PROPOSED REDEPLOYMENT MODELS

Previous research has proposed a method that uses DSE and SMT solvers to solve the optimal redeployment scheme. We propose new redeployment models based on the constraints and objectives based on the work [27]. We argue that different redeployment models for the same constraint or objective can significantly influence the final optimization results and scalability. Redeployment models that have not been properly designed have poor scalability, resulting in a redeployment architecture that can only be used for simple control applications. A well-designed redeployment model can produce highly accurate results and improve the scalability and automation of the entire architecture. A total of four redeployment models have been designed to investigate four different requirements for redeployment. The four conditions are:

- A finite number of available IEC 61499 runtime docker containers.
- A maximum number of FBs per container.
- Minimizing data connections across containers.
- Matching the required skills, e.g., I/O capability, of FBs with the supported skills of containers. The configuration of container skills is initialized via equations (7), (8) and (9) as meta information.

For each function block, a variable x_i is created, and the value of each variable represents the container tag $(1, 2, ..., ContainerNum)$. For example, deploying function block x_i to container with tag "1" indicates $x_i = 1$. Finally, the redeployment result calculated with the Z3 solver is an assignment to all FBs x_i in a control application. Therefore, the assigned value for each function block should not be greater than $ContainerNum$. The redeployment model for this constraint is shown in (1).

$$\begin{aligned} &\forall x_i \in FunctionBlocks, \\ &1 \leq x_i \leq ContainerNum \end{aligned} \tag{1}$$

In addition to the fact that the assignment to each function block variable cannot exceed the number of containers, we impose a second constraint. This restriction is that there is an upper limit to the number of FBs that each container can hold, i.e., the number of variables assigned the same value cannot exceed this limit. The redeployment model for this limit is shown in (2). For example, if the upper limit is two FBs per container, 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.

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

The third formula (3) expresses 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 so that the total number of cross-container data connections is minimized, based on all the above constraints. Suppose two FBs that need to communicate are assigned to two different containers. In that case, they need to be externally communicated via additional service interface FBs. Therefore, an intensity matrix is used to represent external communication stress. This matrix refers to the number of data connections between each pair of FBs. So what the third formula means is to minimize the sum of all FBs external data communications.

$$\begin{aligned} &\forall x_i, x_j \in FunctionBlocks, \\ &\min \sum_{x_i \neq x_j} intensity\,[i]\,[j] \end{aligned} \tag{3}$$

The fourth formula (4) indicates that the skills required for each FB should match the skills supported by the assigned container. The swSkill and hwSkill are artifacts/variables representing the hardware required by FBs and hardware supported by assigned containers/devices. In this paper, we define that $swSkill\,(x_i) = 1$ indicates the x_i needs to access the I/O to work properly. Likewise, $hwSkill\,(1) = 1$ means container1 supports I/O reading and writing for FBs running on it.

For example, an FB that requires Input/Output(I/O) support needs to be deployed to a container with I/O support. Otherwise, FB can not read or write data using the container's I/O successfully. Equations 1, 2, and 3 illustrate how to minimize data communication across devices on top of existing infrastructure. The introduction of the fourth formula further

enhances the rationale for the entire redeployment. We have to consider the relationship between FB and the container or device when recalculating the optimal deployment. This is the only way to ensure the automation, reliability, and efficiency of the entire redeployment process.

$$\begin{gathered}\forall x_i \in FunctionBlocks \\ Containers(1,2,3)\ support\ I/O\ skill, \\ (swSkill\,(x_i) > 0) \Rightarrow \\ (((x_i = 1)\ \&\&\ (swSkill\,(x_i) = hwSkill\,(1)) = true)\ || \\ ((x_i = 2)\ \&\&\ (swSkill\,(x_i) = hwSkill\,(2)) = true)\ || \\ ((x_i = 3)\ \&\&\ (swSkill\,(x_i) = hwSkill\,(3)) = true)) \end{gathered} \quad (4)$$

V. EVALUATION

A. TEST CASE

Fig. 2 shows the test case used in this experiment. The ten FBs make up a simple application. This control application implements a parallel FlipFlop function where the signals received by IX from pin 2 of I/O are passed to output I/O ports 1, 3, 5, and 7 of QX, QX1, QX2, and QX3, respectively. The test case is initially deployed with all the FBs deployed on the same container or device. We then assume that the application is now distributed across multiple available containers, using the constraints and optimization objectives mentioned earlier. Based on these constraints, we propose two optimization objectives. The first objective is to minimize communication across all containers, thereby reducing processing latency and device communication pressure. The second objective is to deploy FBs into containers that meet the requirements, e.g., if a particular FB requires the container's I/O functionality, how to meet this requirement on top of satisfying all the constraints.

For the second optimization objective, compatibility between optimization objectives must be considered when more than one optimization objective needs to be satisfied at the same time. An example of such compatibility is considering the priority of the optimization objectives when both objectives need to be satisfied. When different priorities are applied to different cases, we will get different optimization results. For these two optimization objectives (minimizing external communication and satisfying the skill demand of FBs), the priority is to satisfy the skill demand of FBs first, i.e., to allocate the I/O-demanding FBs (IX and QX FBs) first and then to allocate the other FBs. The redeployment calculation experiment is based on a computer with the 2.6GHz Intel Core i7 processor and 8GB RAM.

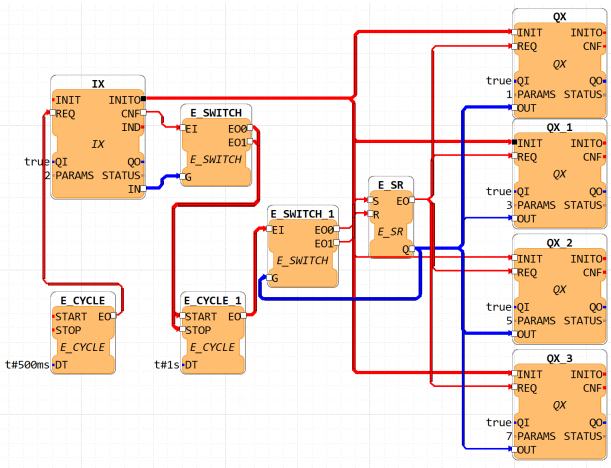

Fig. 2. A simple FlipFlop 4diac application.

B. INITIALIZATION

In order to perform the optimization calculation, besides metadata of the control applications, some constraints should also be initialized with developer-defined values. The number of containers and the maximum number of FBs each container can support is not the metadata obtained by the Parser from the .sys file. Therefore, the value of these two constraints is defined by the developer.

Equations (5) and (6) specify that the number of currently available containers is five, and the maximum capacity of each container is five. Equations (7), (8) and (9) indicate that among the five containers only container 1, 2 and 3 support I/O functionality, while the rest of the containers have $hwSkill(container*) = 0$. Similar to $hwSkill()$, there is a similar FBs hashMap, which is $swSkill()$. If there is FB x1, and there is a demand for I/O skill, then there is $swSkill(x1) = 1$.

$$numMaxFBs = 5 \quad (5)$$

$$numOfContainers = 5 \quad (6)$$

$$hwskill\,(container1) = 1 \quad (7)$$

$$hwskill\,(container2) = 1 \quad (8)$$

$$hwskill\,(container3) = 1 \quad (9)$$

C. RESULTS

As shown in Fig. 2, there are ten FBs in total, and five of them require the I/O functionality of the containers. Therefore, redeployment should first satisfy the needs of these five FBs for the underlying skill requirement because deploying any of them to a container that does not support I/O is unreasonable, and the application will not work properly. For the experiment, we expect a redeployment that minimizes external data connections between the FBs based on the fact that all five FBs can be assigned to containers that support I/O functionality. Fig. 3 shows the current optimal redeployment results calculated by the Z3 solver for the above optimization models. The result shows that a value from 1 to 3 out of 5 is assigned for each FB, which indicates the container label to which the FB should be deployed. From this result, it can be seen that FBs (E_SR, QX, QX_1, QX_2, and E_SWITCH_1) are recommended to be deployed in the same container (container3). Since there are data connections between E_SR and each QX* and E_SWITCH_1, respectively, these six FBs should be deployed in the same container to minimize the device-crossing data connections. However, since redeployment sets a limit of five FBXs per container, one of

```
(define-fun x10 () Int    ;x10=QX_3
  2)
(define-fun x9 () Int     ;x9=QX_2
  3)
(define-fun x8 () Int     ;x8=QX_1
  3)
(define-fun x7 () Int     ;x7=QX
  3)
(define-fun x6 () Int     ;x6=E_SR
  3)
(define-fun x5 () Int     ;x5=E_SWITCH_1
  3)
(define-fun x4 () Int     ;x4=E_CYCLE_1
  2)
(define-fun x3 () Int     ;x3=E_SWITCH
  2)
(define-fun x2 () Int     ;x2=IX
  2)
(define-fun x1 () Int     ;x1=E_CYCLE
  1)
(define-fun MinSumOfIntensity () Int
  1)
```

Fig. 3. The result from the Z3 solver for minimizing the sum of external data connections.

these six FBXs must be deployed to another container (QX_3 is deployed to container2 indicated by the result). Therefore, it can be seen that, based on the initial constraints, Eq. (5) (6) (7) (8) (9), Z3 solver gives the result shown in Fig. 3 as the current optimal redeployment result.

D. DEPLOYMENT

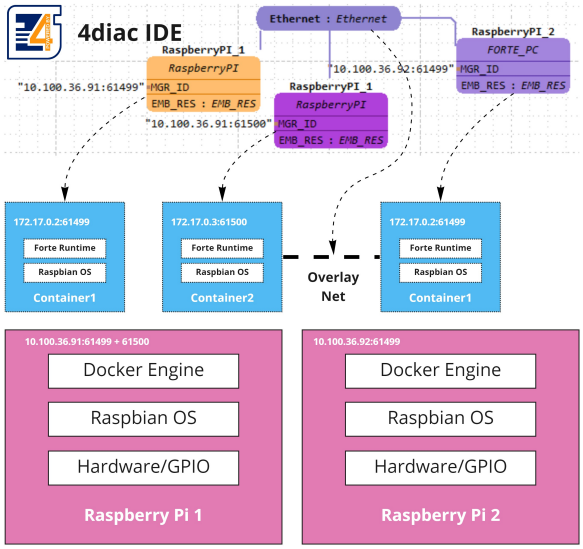

Fig. 4. The deployment for FBs from 4Diac to containers.

Once the deployment optimization results are obtained from the Z3 solver, all FBs need to be deployed to the corresponding containers. The first step in deployment is to containerize the 4Diac runtime (Forte). By deploying the runtime containers, the FBs can work properly inside containers. After the required number of runtime containers are built, the corresponding devices should be created in the 4Diac IDE. These devices should be mapped to the created runtime containers through IP address and port number. The Fig. 4 gives the deployment method for FBs.

TABLE I
THE EXECUTION TIME OF TWO SCENARIOS IN ms.

#execs.	Scenario1			Scenario2		
	10FBs	15FBs	20FBs	10FBs	15FBs	20FBs
1	155	814	541872	150	782	26431
2	152	729	542383	150	747	26268
3	159	750	515431	151	761	26566
4	158	756	527317	150	740	24361
5	158	744	546159	153	735	24128
6	162	781	543024	168	793	24685
7	161	966	587496	160	770	26527
8	153	788	548589	171	755	24133
9	154	779	548789	157	767	29667
10	154	790	558473	162	770	25743
Avg	156.6	789.7	545953.3	157.2	762	25850.9

E. SCALABILITY

For the control application is shown in Figure 1 with only 10 FBs, the proposed redeployment models and implementation of the redeployment framework can achieve the optimal redeployment results. However, for the redeployment problem, in addition to the optimality of the result, the runtime capability should be considered. Therefore, we conducted a controlled experiment for different complexity applications to compare the impact of a varying number of FBs on the redeployment execution delay. The system can run up to 25 FBs (numMaxFBs*numOfContainers) while still using the previous initialization constraints, but we will keep the maximum number as 20 FBs.

Using the same initial constraints and the same redeployment models, the redeployment calculation for four different applications can be performed to obtain their respective calculation delays. In addition to the first application shown in Figure 1, the other two applications are obtained by increasing the number of QX and non-I/O required FBs (scenario1 and scenario2), respectively.

Table I compares the calculation delay for two scenarios. It can be seen that as the number of QXs increases (scenario1), the time spent on redeployment will increase dramatically. This significant increment is because increasing the number of QXs leads to significantly increasing the time it takes to match between $swSkill()$ and $hwSkill()$. However, when increasing the number of other FBs (scenario2) that do not require I/O, the execution time increases slowly compared to the scenario1 due to less calculation in the skill-matching step with the equation (4).

VI. CONCLUSION AND FUTURE WORK

Flexible and dynamic redeployment has always been a crucial and challenging topic. With the increasing demand for smart factories and customized services, the ability to dynamically reconfigure will largely determine the cost of failure recovery and the ability to plan multi-product dynamics. Based on our implemented Z3-based redeployment software and redeployment models, redeployment optimization

results can be obtained in almost runtime for simple control applications. We also present experiment results for scalability to more complex applications with measurement of execution delay. The proposed architecture and redeployment models are more scalable than the results obtained in other related studies.

In future research, more redeployment models will be presented to describe more complex redeployment problems. We will also apply upgraded Z3 solver on more complex and IEC 61499-relevant applications. In addition, the Z3-based redeployment software needs to be further optimized together with the redeployment models to improve the runtime performance and flexibility of the redeployment calculation.

REFERENCES

[1] X. Wei, S. Yuan, and Y. Ye. Optimizing facility layout planning for reconfigurable manufacturing system based on chaos genetic algorithm. *Production & Manufacturing Research*, 7(1):109–124, 2019.

[2] G. Büchi, M. Cugno, and R. Castagnoli. Smart factory performance and industry 4.0. *Technological Forecasting and Social Change*, 150:119790, 2020.

[3] 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.

[4] 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.

[5] IEC 61499. Function blocks–Part 1: architecture, second edition. *International Electrotechnical Commission, Geneva, Switzerland*, 2012.

[6] V. Vyatkin. Iec 61499 function blocks for embedded and distributed control systems design. 2015. 3rd edition.

[7] C. Barrett, R. Sebastiani, S. A Seshia, and C. Tinelli. Satisfiability modulo theories.

[8] H. Manninen, V. Jääskeläinen, and J. O. 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.

[9] 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.

[10] 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.

[11] 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.

[12] L. De M. and N. Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[13] T. Strasser, M. Rooker, G. Ebenhofer, A. Zoitl, C. Sunder, A. Valentini, and A. Martel. Framework for distributed industrial automation and control (4diac). In *2008 6th IEEE International Conference on Industrial Informatics*, pages 283–288. IEEE, 2008.

[14] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, and I. Meedeniya. Software architecture optimization methods: A systematic literature review. *IEEE Transactions on Software Engineering*, 39(5):658–683, 2012.

[15] S. Zverlov, M. Khalil, and M. Chaudhary. Pareto-efficient deployment synthesis for safety-critical applications in seamless model-based development. 2016.

[16] J. Eder, S. Zverlov, S. Voss, M. Khalil, and A. Ipatiov. Bringing dse to life: exploring the design space of an industrial automotive use case. In *2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 270–280. IEEE, 2017.

[17] A. Diewald, S. Voss, and S. Barner. A lightweight design space exploration and optimization language. In *Proceedings of the 19th International Workshop on Software and Compilers for Embedded Systems*, pages 190–193, 2016.

[18] V. Marik and D. McFarlane. Industrial adoption of agent-based technologies. *IEEE Intelligent Systems*, 20(1):27–35, 2005.

[19] P. Leitão. Agent-based distributed manufacturing control: A state-of-the-art survey. *Engineering Applications of Artificial Intelligence*, 22(7):979–991, 2009.

[20] P. Leitão, A. W. Colombo, and S. Karnouskos. Industrial automation based on cyber-physical systems technologies: Prototype implementations and challenges. *Computers in industry*, 81:11–25, 2016.

[21] C. Savaglio, M. Ganzha, M. Paprzycki, C. Bădică, M. Ivanović, and G. Fortino. Agent-based internet of things: State-of-the-art and research challenges. *Future Generation Computer Systems*, 102:1038–1053, 2020.

[22] S. Karnouskos, P. Leitao, L. Ribeiro, and A. W. Colombo. Industrial agents as a key enabler for realizing industrial cyber-physical systems: Multiagent systems entering industry 4.0. *IEEE Industrial Electronics Magazine*, 14(3):18–32, 2020.

[23] J. Yan, C.g Pang, C. Yang, and V. Vyatkin. Adaptable software components: Towards digital ecosystems and software evolution in the industrial automation domain. In *IECON 2014-40th Annual Conference of the IEEE Industrial Electronics Society*, pages 2512–2518. IEEE, 2014.

[24] T. Goldschmidt, S. Hauck-Stattelmann, S. Malakuti, and S. Grüner. Container-based architecture for flexible industrial control applications. *Journal of Systems Architecture*, 84:28–36, 2018.

[25] R. Sinha, K. Johnson, and R. Calinescu. A scalable approach for reconfiguring evolving industrial control systems. In *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, pages 1–8. IEEE, 2014.

[26] T. Terzimehic, S. Voss, and M. Wenger. Using design space exploration to calculate deployment configurations of iec 61499-based systems. In *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, pages 881–886. IEEE, 2018.

[27] T. Terzimehić, M. Wenger, S. Voss, S. Grüner, and H. Elfaham. Smt-based deployment calculation in industrial automation domain. In *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 290–297. IEEE, 2019.

[28] P. Dehraj and A. Sharma. A review on architecture and models for autonomic software systems. *The Journal of Supercomputing*, 77(1):388–417, 2021.