Programming Abstractions for Simulation and Testing on Smart Manufacturing Systems

Chiao Hsieh, Daniel Wu, Yubin Koh, Sayan Mitra
University of Illinois at Urbana-Champaign
Urbana-Champaign, IL, USA
{chsieh16,dxwu2,yubink2,mitras}@illinois.edu

***Abstract*— A smart manufacturing system is a complex cyber-physical system consisting of a collection of component machines and a floorplan layout defining the spatial relationship between components. Each component may be of different physical behavior with different control software. Simulation and testing on smart manufacturing systems require a software infrastructure that can orchestrate the execution of heterogenous, cyber-physical components besides modeling physical machines in respect to floorplan layouts. Automated simulation as a result is challenging and error-prone. Recent strides in formal modeling of cyber-physical systems and programming languages offer some new techniques for addressing this challenge. In this paper, we present a compositional automata-based modeling formalism and programming abstractions to design coordination logic between heterogeneous robots in different layouts. Our formalism allows us to automatically simulate and compare performance metrics for different floorplan layouts. We implement our proof-of-concept prototype with the challenging simulation environment for 2021 Agile Robotics for Industrial Automation Competition. Our experiment results demonstrate how our simulation can be used to evaluate and compare performance under different layouts and applicable for reconfiguration and virtual commissioning.**

***Index Terms*— smart manufacturing, virtual commissioning, simulation**

I. Introduction

Modeling and simulation are essential for rapid testing and debugging of smart manufacturing systems. For example, in *virtual commissioning*, many different machine configurations, layouts, and variations of associated control programs have to be evaluated, before the actual system is commissioned. The state of the art and the outstanding challenges surveyed in [1]–[3] suggest that the high level of expertise and effort required for creating such simulation models make virtual commissioning prohibitively expensive, especially for small and medium-sized enterprises. The challenges arise from two distinct sources. First, developing digital models of physical machines and mechanisms is challenging and requires domain expertise. And second, even with available component models, the software infrastructure that can orchestrate the execution of heterogeneous, cyber-physical components in a manageable simulation is challenging and error-prone. Recent advances in formal modeling of cyber-physical systems [4]–[6] and programming languages, offer some new techniques for addressing this second challenge. These techniques not only provide a sound mathematical basis for developing complex models, but they also offer software for creating executable simulation programs [7], [8]. In this paper, we propose an approach that builds-up on these recent advances to show how the burden of developing simulation models can be reduced with abstractions and compositional modeling.

This project is supported by NSF-CNS Award# 1544901.

Broadly, a simulation model for a manufacturing system consists of a collection of component machines (we call them generically as *robots* in this paper) and a floorplan layout that defines the spatial relationship between the robots. Robots may be of different types, they may have different control software, and they interact physically (e.g., through transfer of physical materials) and logically (e.g., through transfer of data as needed by a coordination software layer on top of control software). A simulation framework for a manufacturing system therefore requires a good abstraction for (i) modeling physical interactions such as reading sensor and writing actuator ports on the robot, (ii) programming coordination software layer such as accessing shared variables between control software (iii) composing all robots, control software, and coordination software layer while considering the floorplan layout.

In recent years, thanks to the effort from government and the open source community, the Agile Robotics for Industrial Automation Competition (ARIAC) [9], hosted by the National Institute of Standards and Technology (NIST), provides freely available simulated industrial robots such as the robot arms from Universal Robots and rail-guided vehicles in Figure 1. These high-fidelity simulation frameworks enable exploring different combinations of available robot models to improve smart manufacturing systems. For example, using less robots can help reduce cost and lower redundancy in general. With more robots, there will be higher cost, but the performance is not guaranteed to improve and depends on good coordination of control software. Testing and evaluating the simulation can thus help to answer if a new layout with more robots can achieve higher throughput or only creates redundancy. However, running the simulation and evaluating the performance remains error prone because the current simulation frameworks have not addressed the complexity of coordination between the control software and the composition of heterogeneous robots under different layouts.

In this work, we demonstrate the feasibility of automated simulation of different layouts of robot placements to eval-

Fig. 1. GEAR simulation environment for the ARIAC 2021 competition.

uate performance, throughput, and cost-effectiveness. Given a valid layout, we showcase how to automatically generate and execute the parameterized controller to obtain analytic from simulations. Our framework therefore allows users to explore the improvement and trade-offs in the design of smart manufacturing systems.

In summary, the contributions in this paper are as follows: (a) We present a compositional automata-based modeling formalism to program coordination logic between heterogeneous robots in different layouts. (b) We propose an analysis and simulation framework for our modeling formalism to automatically measure and compare performance metrics for different layouts. (c) We implement and demonstrate our proof-of-concept prototype with the high-fidelity simulation environment in ARIAC 2021.

Looking ahead, we believe that the type of compositional modeling illustrated in this paper can not only lower the barrier to creating simulation models for smart manufacturing systems, but it can also help with developing and tuning controller programs, performance evaluation [10], and anomaly detection.

Related works: Research on software defined control (SDC) framework [11]–[13] has proposed approaches on the management and monitoring for smart manufacturing systems and provides a global view of the system for decision making. All of the works benefit from integrating fixed simulation models or digital twins for predictive decision making and anomaly detection. Our simulation approach with parameterized floorplan layouts can further extend the existing simulation for more advanced application such as virtual commissioning and reconfiguration.

A number of ideas from the extensive body of research on modeling for cyber-physical systems inform our modeling formalism (see [4]–[6], [14] and the citations therein). A previous mathematical formalism (without realistic simulations) for smart manufacturing appeared in [15]. The formalism presented in this paper integrates the notion of shared memory and controller ports from our Koord [7] programming language.

II. Motivating Example

We illustrate our motivating example following the Agile Robotics for Industrial Automation Competition (ARIAC) [9] hosted by the National Institute of Standards and Technology (NIST). In particular, we study ARIAC 2021: a smart manufacturing scenario revolving around the theme of the COVID-19 pandemic along with the Gazebo Environment for Agile Robotics (GEAR).

GEAR for ARIAC 2021: In ARIAC 2021, the goal is to design the software for a smart manufacturing system that will assemble on-demand, ventilator briefcases consisting of four items: a battery, a sensor, a regulator, and a pump. The competing control software is required to transport the correct items from the conveyor belt to one of the assembly stations (`as1` to `as4`) and assemble the briefcase as shown in Figure 1. GEAR consists of four types of robot platforms: a conveyor belt, a kitting robot, Automated Guided Vehicles (AGVs), and a gantry robot. The conveyor belt serves as an entry point for new items and moves items from left to right. The kitting robot consists of a UR10 robot arm, a vacuum gripper to grab items, and a linear rail to move parallel to the conveyor. The AGVs carry items along a defined path with fixed destinations and deliver items to assembly stations. Lastly, the gantry robot consists of another UR10 robot arm that is attached to a rotatable torso, which can move along a two-dimensional plane on two linear rails.

Simulations for layout optimization: A crucial application of the simulation environment is to evaluate over different layouts of robots to compare various metrics such as throughput, safety, robustness, etc. This will allow finding the optimized layout. However, automated simulation for different layouts has to address two concerns: (i) the controllers for robots will require adjustments under different layouts, and (ii) the physical parameters, such as the operating range of the robot arms, limit the relative placements of the AGVs.

To address the first concern, our approach of parameterized automata with shared variables generalizes the controllers to work under different floorplan layouts. More specifically, we are able to support all four main robot types given in ARIAC 2021 the conveyor belt, AGVs, the kitting robot, and the gantry robot. An automaton is constructed for each of the existing robots so that it can accordingly carry out its own tasks, and the composition of all automata is used to simulate the end to end goal of delivering items between the robots and assembling the ventilator.

To solve the second concern, we consider the information provided in the floorplan layout. For the different layouts shown in Figure 2, the problematic layout on the right does not consider the operating range of robot arm, and hence the robot arm cannot transfer items to the AGV. We consider this as an infeasible layout with a *connectivity* issue. Our approach takes in different layouts as inputs and generates a *connectivity graph* to ensure the connectivity of robots, such as the one shown in Figure 3. If there exists a connected

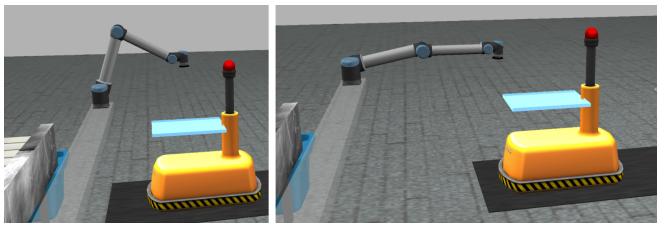

Fig. 2. Feasible vs infeasible layouts constrained by the operating range of the robot arms. The robot arm is able to reach the AGV in a feasible layout (Left). The robot arm is too far from the AGV in an infeasible layout (Right).

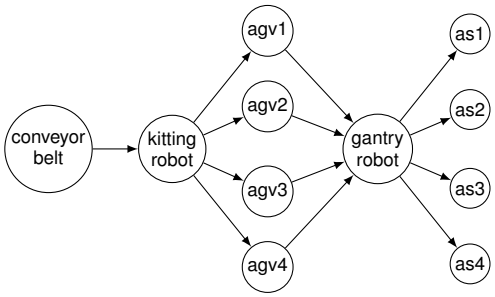

Fig. 3. The connectivity graph generated for the default world in Figure 1.

path from the source to the destination in our connectivity graph, items can be transferred from the conveyor belt to the assembly stations. In other words, we can detect the connectivity issues by checking if a path exists from the source to the destination, which then can be used to exclude infeasible layouts, since the robots will not be able to work together to transfer the items. We will formally define the connectivity graph and check connectivity using standard graph search algorithms in Section III-A.

III. Compositional Modeling of a Heterogeneous Multi-Agent Systems

In this section, we will discuss our compositional modeling framework for smart manufacturing systems. We will show how each instance of a robotic platform is modeled as a special type of a *parameterized automaton*. Each type of automaton has different sets of *shared variables* for communication with other automata and *ports* to interact with the physical environment. We then discuss the definition of the complete system as a composition of various automata instances defined by a *layout*. This compositional definition enables us to both automatically check the compatibility of a layout and generate executable simulation code.

Robot components: The overall system is built by instantiating a number of robots from a library of possible robot types. Let the set of robot types be $\{R_1, R_2, \ldots, R_N\}$. For example, R_1 = AGV and R_2 = Conveyor. A system consists of robot platform instances $\{r_1, ..., r_k\}$ of different robot types. If a robot platform instance r_i is of the robot type R_j, we denote it as $type(r_i) = R_j$. The robot instance agv1 in Figure 1 is of the type AGV. We can write it as $type(\text{agv1}) = \text{AGV}$.

Each type of the robots can be customized or instantiated by fixing a number of *robot parameters*, such as location, orientation, and certain controller parameters. For each robot type R_i, we write the corresponding parameter space as P_{R_i}. For example, agv1 and agv2 in Figure 1 are both instances of AGV with different station positions inside the warehouse. Assuming there are three required parameters, the position of the start station, the interval between the stations, and the number of items to transfer in each trip, the parameter space for AGV is $P_{\text{AGV}} = \mathbb{R}^2 \times \mathbb{R} \times \mathbb{N}$. The particular parameter values of agv1 is denoted as $param(\text{agv1}) = ((-2.3, 4.6), 0.73, 4) \in P_{\text{AGV}}$ to represent that the start station is placed at (-2.3, 4.6), the interval between stations is 0.73 meters. Or equivalently, it defines the station positions at (-2.3, 4.6), (-3.03, 4.6), and (-3.76, 4.6), and agv1 transfers 4 items for each trip.

System Composition: Our formalization is inspired by the transition system model for software defined controllers in [15] and the shared memory model for distributed robotics system in [7]. We model the entire system as the composition of all robot instances:

$$Sys \triangleq \mathcal{A}(r_1) \parallel \mathcal{A}(r_2) \parallel ... \parallel \mathcal{A}(r_k)$$

where each robot instance is a discrete automaton $\mathcal{A}(r) = (X, Q, \Theta, \Sigma_r, \delta)$ where (i) $X = X^L \dot{\cup} X^G \dot{\cup} X^P$ is a finite set of *state variable* names. X^L, X^G, and X^P denotes the sets of *local* variables, *shared* variables, and *controller port* names respectively. We assume the function $type(x)$ to return the set of possible values for x. A *state* q is a mapping from $x \in X$ to a value $q(x) \in type(x)$ and $val(X)$ denotes the set of all possible states of X. (ii) $Q \subseteq val(X)$ is the set of states. (iii) $\Theta \subseteq Q$ is the set of initial states. (iv) Σ_r is the set of all actions. The actions are parametrized by the robot parameters $param(r)$. (v) $\delta \subseteq Q \times \Sigma_r \times Q$ is the transition relation.

Figure 4 demonstrates how the local variable (variable s to represent the current status), shared variables (all_loaded and all_dropped), controller ports (pos), and robot parameters are used to define the (partial) automaton for agv1. The system starts from LOADING status and waits for the product items being loaded on the tray. It waits for the shared variable all_loaded to become true before entering MOVING status. This shared variable may be controlled by other robot instances, such as the kitting arm robot, to allow loading and delivering multiple items at a time. Notice that the "goto" action is parametrized by the position of the station at (-3.03,4.6). We can generalize the action to support the other station at (-3.76, 4.6). Once entering the MOVING status, it may take multiple cycles to reach to the particular station. In this case, the value of controller port pos is decided and changed according to the physical environment instead; therefore, it monitors the controller port variable pos until its position reaches the station at (-3.03,4.6) and transits to the DROPPING status. Similarly, all_dropped shared variable is used to decide whether all items have been dropped to the assembly station so that it can enter RETURNING status to go back to the starting station and reset to LOADING status.

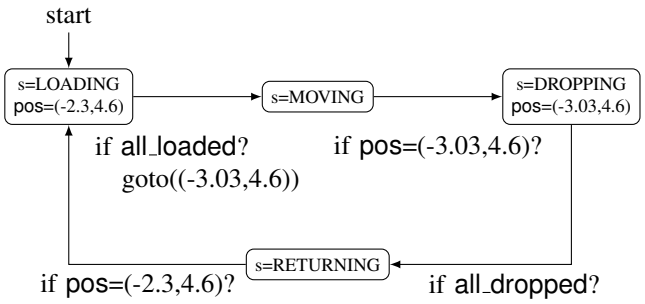

Fig. 4. Partial automaton for the robot instance agv1 working between stations at (-2.3,4.6) and (-3.03,4.6).

A. Connectivity check for feasible layouts

For simplicity, we consider the robot instances as places in the two-dimensional space $\mathbb{R}^2$. We begin by computing the operating ranges of all the robots specified in a layout. Figure 5 demonstrates how the position, orientation, and range parameters in the environment is specified in JSON format, and the annotated operating ranges in the Gazebo simulation environment. The type of robot determines the operating range of motion. For instance, the kitting robot's range of motion is a cylinder with its axis along the x or y-direction. An AGV's operating range can be represented by a set of 2D vertical lines, since other robots will be dropping items and grabbing them off of the AGV. Since computing the geometric space of the operating range are standard, we skip the detail of the computation and use $range(r_i)$ to denote the operating range derived from a given layout.

Formally, a connectivity graph is a directed graph $G = \langle V, E, s, T \rangle$ where V is the set of all robot instances $\{r_1, r_2, ..., r_k\}$, an edge $(r_i, r_j) \in E$ represents that an item can be directly transferred from r_i to r_j, $s \in V$ is the source node, and $T \subset V$ is the set of destination nodes. The process of generating a connectivity graph is as follows. Starting from a graph with all robot instances $\{r_1, r_2, ..., r_k\}$, but without any edges, we mark the robot instance where the items enters the system as the source node s and the robot instances where the final product item leaves the system as the destination nodes $t \in T$. For the connectivity graph in Figure 3 as an example, the conveyor belt robot is the source node $s =$ conveyor, and the assembly stations are the destination nodes $T = \{\text{as1}, \text{as2}, \text{as3}, \text{as4}\}$. We add an edge from node r_i to node r_j if and only if (i) the operating ranges of robot instances intersect, i.e., $range(r_i) \cap range(r_j) \neq \emptyset$ and (ii) robot r_i would transfer an item over to robot r_j according to the robot types $type(r_i)$ and $type(r_j)$.

Finally, we can check if every node is reachable starting from the source node s, and every node can reach any destination node $t \in T$. A standard algorithm is to use depth-first search (DFS) starting from the source node s. If there is no path from the source node s to a node r_i, then there is no path for an item to be delivered to r_i, hence the robot instance r_i is not usable. Similarly, if there is no path from r_i to any destination node t, the item will never be used in the final product. We can therefore detect and reject these invalid layouts by constructing the connectivity graph.

IV. Implementation

In this section, we present our method of implementation for building our system. The three major components of this include (1) the input floorplan layout in the form of a JSON file and the generation of a Gazebo world as an SDF file, (2) controller automata, and (3) a connectivity check. All of our implementation is in an open source repository at the link below. [16]

Programmable layouts to Gazebo environments: We parse the input JSON file and retrieve all dynamic (robots) and static (assembly stations, bins, other obstacles). We then write this information in world files that are applied to SDF files when the Gazebo world is launched. The resulting world should show the floorplan layout specified in the JSON file as a Gazebo simulation.

Building robot controllers: As discussed in Section III, each robot follows an automata depending on the robot type with tunable robot parameters. To implement the automaton, the basis of all robotic movement in our simulation is done through ROS, specifically through messages passed over ROS topics and ROS service calls. For simpler robots, such as the conveyor belt and the AGVs, a simple command to control the speed is enough. For more complex robot arms such as the kitting and gantry robot, we use the MoveIt framework for sending ROS messages to control the arm joints. On top of MoveIt, we use an analytical-based inverse kinematics approach for motion planning to generate a sequence of desired arm joint values and brings the vacuum gripper to a desired position.

The MoveIt framework provide well established motion planning tool that allowed us to abstract over the lower level details of sending ROS messages. This helped us build an analytical approach-based inverse kinematics solver for the two robot arms without going into an extreme level of detail. There are also plenty of alternatives including sampling-based motion planner, such as a Probabilistic Roadmap (PRM) or a Rapidly-Exploring Random Tree (RRT).

Connectivity Check: We use a short Python function to implement our connectivity check. In short, we create a node for each robot specified in the JSON file and then create a directed edge between nodes where robots can transfer items to one another. We implement a standard depth-first search algorithm and run it on the starting node to determine whether connectivity from start to finish is satisfied or not.

V. Experiment Results

We use the competition interface provided by the ARIAC for running our experiment. Items will be spawned on the conveyor belt and shipped to briefcases located at assembly stations. We run the simulations on three different layouts, namely the default competition layout in Figure 1 (Default), a more spaced layout (Spaced), and a layout with less AGVs (LessAGV):

```
"robots": [
    {   "name": "kitting",
        "type": "kitting",
        "pose": [-1.3, 0, 1.127],
        "rail_dim": "y",
        "rail_range": [-4.8, 4.8]},
    {   "name": "conveyor_belt",
        "type": "conveyor",
        "pose": [-0.573, 0, 0],
        "orient": "y",
        "orient_range": [-5,5]}
]
```

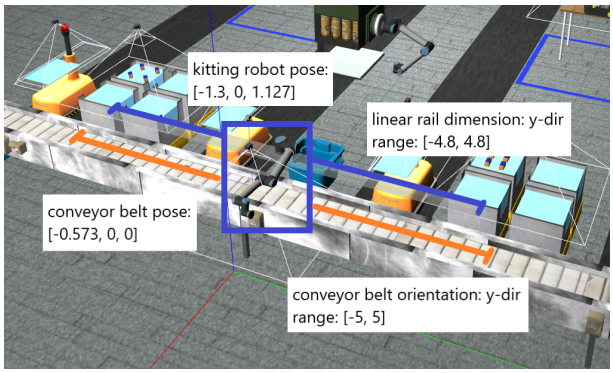

Fig. 5. A programmable layout in JSON format (*Left*) and the corresponding Gazebo simulation environment (*Right*).

- Default layout has one conveyor belt, one kitting robot, four AGVs, and one gantry robot. The default space between the conveyor belt and the kitting robot is 0.73 meter. The starting poses of the four AGVs (in order from 1 to 4) are (-2.3, 4.6), (-2.3, 1.3), (-2.3, -1.3), and (-2.3, -4.6).
- Spaced layout sees increased space between the conveyor belt and kitting robot to 0.93, as well as increased space between the four AGVs. The starting poses of the AGVs are now (-2.5, 4.2), (-2.5, 0.9), (-2.5, -0.9), (-2.5, -4.2). It uses the same number of robots as the Default layout.
- LessAGV world has the same layout with the same spacing as Default world, with the removals of two robots: agv2 and agv4.

We also consider varying the robot parameters for the robot automata. The tunable robot parameter in our experiments is the capacity of an AGV, the maximum number of items loaded for each trip from the conveyor belt to assembly stations, denoted as *Cap*.

Recall from Section II that an order is to assemble a ventilator briefcase composed of four different kinds of items. For each combination of a layout and a parameter value, we conduct a simulation of filling 8 orders and record the total time usage of an simulation run. We then divide the total time by 8 to calculate the average amount of time for finishing one order as shown in Table I.

All of our experiments are conducted on a Ubuntu workstation (version 18.04.6 LTS) with CPU model Intel Xeon Silver 4110, GPU model NVIDIA Corporation GP104GL, ROS Melodic (Version 1.14.12), and Gazebo 9.16.0.

TABLE I

AVERAGE TIME TO FINISH ONE ORDER IN SECONDS UNDER DIFFERENT COMBINATIONS OF SIMULATION OF SHIPPING 8 ORDERS.

	Cap = 2	*Cap* = 4	*Cap* = 8
Default	196.650	173.037	133.327
Spaced	198.654	197.384	156.379
LessAGV	TIMEOUT	180.993	222.375

Table I shows that the time usage is generally worse with increased space, but only marginally for certain values of *Cap*. We see significant improvements with loading more items on less AGVs for the default world and the spaced world, suggesting that operations with the gantry robot are the bottleneck and that optimizations regarding the gantry movement and its parameter tuning should be the focus in future runs. In LessAGV layout with less AGVs, we see significantly higher times to ship items but a different parameter value that produces optimal throughput. The results here show the possible improvements by adjusting just one robot parameter.

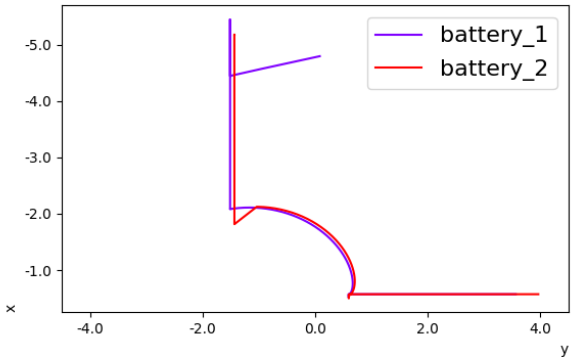

Fig. 6. 2D position trajectories of two battery items in Default layout. XY-axes are rotated 270° to match Figure 1. The trajectories start from the lower right corner. Both batteries are moved to left by the conveyor belt. put on agv3 separately by the kitting_robot, and carried to the assembly station as3 in one trip. Finally, battery_1 is transferred to the assembly table by the gantry_robot.

In addition, we track the trajectories of items in transitions for runtime monitoring. Figure 6 shows the position trajectories of two battery items starting from the right end of the convey belt, i.e., lower right of the plot. Both batteries moves to left by the conveyor belt. The curves denote the two items are transferred to agv3 at (-2.3, -1.3) by the kitting_robot, put down slightly separated, and carried to the assembly station as3 in the same trip. Finally, the plot shows the gantry_robot picked up and transfer battery_1 to the assembly table. This showcases that the prototype is able do fine-grained runtime tracking of individual components. In short, our experiment shows that our framework can provide system-level metrics such as throughput, as well as component-level details such as the trajectories of items, and we can repeat the simulation and analysis for different

layouts and parameters to identify critical components and subsequently optimize the entire smart manufacturing system.

VI. Conclusions

We presented our approach to a compositional modeling and simulation of smart manufacturing systems. Given an input JSON file representing the layout of a manufacturing floorplan, and the parameterized controllers for the robot types, our system can generate a detailed simulator for the entire plant. The solution uses the notion of controller ports and shared variables for physical and logical interaction among robot components. We showed how the generated simulator can be used for measuring and comparing throughput of the ventilator assembly orders in ARIAC 2021 with four distinct robot types. Our current implementation relies on the Gazebo simulator and the ARIAC 2021 scenario; however, the concepts can be implemented on other simulators and scenarios.

Beyond the preliminary experimental results we have discussed, the work suggests several directions for future research. First, the parameterized controller programs used for simulation can be compiled to executable code and deployed on the actual hardware. This can significantly reduce the development and testing cost. A variant of this idea for the Koord language has been demonstrated in the context of mobile robotic applications [8]. Generating PLC code directly from communicating state machine models would be an interesting direction to explore. Second, the compositional models, with the well-defined interfaces (ports and shared variables) should be amenable to automatic generation of runtime monitors [17], [18] for software defined control [11]–[13]. Finally, rapid generation of simulators and their evaluations open-up the possibility of optimizing designs and auto-tuning parameters for manufacturing systems.

Acknowledgment

The authors would like to thank Prof. Kira Barton, Dr. James Moyne, Dr. Yassine Qamsane, Dr. Efe Balta from the University of Michigan, and Prof. Sibin Mohan and Bin-Chou Kao, from University of Illinois for many valuable discussions that informed the research. The research was supported by a research grant from the US National Science Foundation under the CPS Frontiers program (NSF CNS 1544901).

References

[1] S. Süß, S. Magnus, M. Thron, H. Zipper, U. Odefey, V. Fäßler, A. Strahilov, A. Kłodowski, T. Bär, and C. Diedrich, "Test methodology for virtual commissioning based on behaviour simulation of production systems," in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2016, pp. 1–9.

[2] P. Hoffmann, R. Schumann, T. M. Maksoud, and G. C. Premier, "Virtual commissioning of manufacturing systems a review and new approaches for simplification." in *ECMS*. Kuala Lumpur, Malaysia, 2010, pp. 175–181.

[3] C. G. Lee and S. C. Park, "Survey on the virtual commissioning of manufacturing systems," *Journal of Computational Design and Engineering*, vol. 1, no. 3, pp. 213–222, 2014.

[4] R. Alur, *Principles of Cyber-Physical Systems*. The MIT Press, 2015.

[5] S. Mitra, *Verifying Cyber-Physical Systems: A Path to Safe Autonomy*, ser. Cyber Physical Systems Series. Cambridge, MA, USA: MIT Press, Feb. 2021.

[6] N. Lynch, R. Segala, and F. Vaandrager, "Hybrid I/O automata," *Information and Computation*, vol. 185, no. 1, pp. 105–157, 2003.

[7] R. Ghosh, C. Hsieh, S. Misailovic, and S. Mitra, "Koord: A Language for Programming and Verifying Distributed Robotics Application," *Proc. ACM Program. Lang.*, vol. 4, no. OOPSLA, nov 2020. [Online]. Available: https://doi.org/10.1145/3428300

[8] R. Ghosh, J. P. Jansch-Porto, C. Hsieh, A. Gosse, M. Jiang, H. Taylor, P. Du, S. Mitra, and G. Dullerud, "CyPhyHouse: A programming, simulation, and deployment toolchain for heterogeneous distributed coordination," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, May 2020, pp. 6654–6660.

[9] A. Downs, Z. Kootbally, W. Harrison, P. Pilliptchak, B. Antonishek, M. Aksu, C. Schlenoff, and S. K. Gupta, "Assessing industrial robot agility through international competitions," *Robotics and Computer-Integrated Manufacturing*, vol. 70, p. 102113, 2021.

[10] K. An, A. Trewyn, A. Gokhale, and S. Sastry, "Model-driven performance analysis of reconfigurable conveyor systems used in material handling applications," in *2011 IEEE/ACM Second International Conference on Cyber-Physical Systems*, 2011, pp. 141–150.

[11] F. Lopez, Y. Shao, Z. M. Mao, J. Moyne, K. Barton, and D. Tilbury, "A software-defined framework for the integrated management of smart manufacturing systems," *Manufacturing Letters*, vol. 15, pp. 18–21, 2018.

[12] E. C. Balta, D. M. Tilbury, and K. Barton, "A centralized framework for system-level control and management of additive manufacturing fleets," in *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2018, pp. 1071–1078.

[13] Y. Qamsane, C.-Y. Chen, E. C. Balta, B.-C. Kao, S. Mohan, J. Moyne, D. Tilbury, and K. Barton, "A unified digital twin framework for real-time monitoring and evaluation of smart manufacturing systems," in *2019 IEEE 15th international conference on automation science and engineering (CASE)*. IEEE, 2019, pp. 1394–1401.

[14] R. Alur and T. A. Henzinger, "Reactive modules," *Formal methods in system design*, vol. 15, no. 1, pp. 7–48, 1999.

[15] M. Potok, C.-Y. Chen, S. Mitra, and S. Mohan, "SDCWorks: A Formal Framework for Software Defined Control of Smart Manufacturing Systems," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, 2018, pp. 88–97.

[16] "ARIAC repository for CyPhyHouse," https://github.com/cyphyhouse/ARIAC, 2021.

[17] D. Jin, P. O. Meredith, C. Lee, and G. Roşu, "Javamop: Efficient parametric runtime monitoring framework," in *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 2012, pp. 1427–1430.

[18] B. d'Angelo, S. Sankaranarayanan, C. Sánchez, W. Robinson, B. Finkbeiner, H. B. Sipma, S. Mehrotra, and Z. Manna, "Lola: runtime monitoring of synchronous systems," in *12th International Symposium on Temporal Representation and Reasoning (TIME'05)*. IEEE, 2005, pp. 166–174.