Proceedings of the 2021 Winter Simulation Conference
S. Kim, B. Feng, K. Smith, S. Masoud, Z. Zheng, C. Szabo and M. Loper, eds.

DESIGNING AND IMPLEMENTING OPERATIONAL CONTROLLERS FOR A ROBOTIC TOTE CONSOLIDATION CELL SIMULATION

Leon McGinnis
Shannon Buckley
Ali V. Barenji

The Georgia Institute of Technology
School of Industrial and Systems Engineering
Physical Internet Center
Atlanta, GA 30332-0205, USA

ABSTRACT

Operational control is a key driver of production system performance, yet the design of operational controllers is not well-covered in the production systems simulation literature. With a robotic cell consolidating totes for delivery in a logistics hub as the use case, we describe the design of the cell's operational controller and an implementation approach used in an AnyLogic™ hybrid agent-discrete event simulation. Research motivation is discussed. Design principles are clearly explained, and key aspects of implementation for AnyLogic™ are presented. A companion Emulate3D™ model is described for obtaining realistic estimates of operation cycle times. Implications for the engineering of operational controllers in digital twins are addressed.

1 INTRODUCTION

Operational control as defined in the ISA-95 standard (International Society of Automation 2021) is the translation of production plans into the execution of production operations. For example, the plan to produce 100 widgets must be translated into the processes executed by production resources to realize those 100 widgets. Specific production tasks must be identified, tasks must be assigned to specific resources, and queues of tasks must be managed. ISA-95 is the current standard for manufacturing operations management controllers, and while it identifies the information requirements for the various functions in operational control, it does not address the design or implementation of operational controllers; in particular it does not address decision-making in operational control.

Operational controllers are a critical component for smart factories and the realization of initiatives like Industry 4.0, where good reliable operational decision-making is critical to success. Contemporary manufacturing execution systems (MES) are proprietary closed systems, thus are not an adequate foundation for building and testing the conceptual models necessary to develop the computational decision-making bridge from planning to execution that is required for smart manufacturing. Open, generic models and implementation examples are a fundamental requirement.

While there is a vast literature on decision making for operations control functions, such as job scheduling and sequencing, batching, and setup sequencing, this literature is difficult to synthesize with the recommendations of ISA-95 because invariably the research assumes a specific form for the decision problem and focuses on the analysis to resolve that problem. To produce a generic operational controller architecture for ISA-95, attention must be given to the events corresponding to ISA-95's information flows in order to properly identify and define the decision problem relevant for a particular situation.

978-1-6654-3311-2/21/$31.00 ©2021 IEEE

The gap between the research on operational control and decision making and its implementation in smart manufacturing is exemplified in a recent and extensive survey of production planning and control (PPC) in Industry 4.0. Bueno et al. (2020) concluded "no studies were found regarding decision support systems for smart PPC systems, frameworks, or architectures in the context of moving toward PPC digitalization" and recommended research on "development of intelligent decision support systems, frameworks, architectures, and models to advance and consolidate smart manufacturing planning and control".

The contribution of this paper is providing a concrete example of an operational controller that conforms to ISA-95, embodies the necessary computational decision making to be considered "smart" and has a plausible path to implementation. This is accomplished with two different but related models; one based on AnyLogic™ (The AnyLogic Company 2021) to support controller development and application-oriented assessment, and one based on Emulate3D™ (Rockwell Automation 2021) to develop realistic operational cycle times for processes not yet implemented in physical hardware.

Section 2 summarizes the relevant prior literature on the requirements and architecture for manufacturing operational controllers and Section 3 describes a relevant use case. Section 4 addresses the functional design of an operational controller for the use case and Section 5 describes its implementation in AnyLogic™. Section 6 addresses the role of Emulate3D™. Some initial empirical results are discussed in Section 7 and future work is discussed in Section 8.

2 PRIOR RESEARCH

Discrete-event logistics systems or DELS are described by McGinnis (2010). In DELS, flow units with well-defined processing requirements move through a network of resources, each of which has one or more process capabilities. Resources transform the flow units by exercising process capabilities. The unit flow and the execution of resource capabilities are managed by controllers through the assignment of tasks, which are authorizations for an execution of a process capability. The DELS semantics are presented by Sprock et al. (2019) and provide a generic framework within which to explore the realization of operational controllers. Many systems, from warehouses, factories and supply chains to hospitals can be characterized as DELS.

The large literature on scheduling and related issues in manufacturing is examined by Sprock (2016) and a unifying set of operational control decisions is identified. This analysis of operational control is summarized by Sprock et al. (2019), where the complete set of possible operational control decisions are: (1) task accept/reject; (2) task sequencing (time of execution start); (3) task-to-resource assignment; (4) dynamic process planning (typically, identifying material handling tasks); and (5) resource setup (changing resource state). Sprock and McGinnis (2015) propose functional requirements for operational controllers, and McGinnis (2019) proposes the functional controller architecture shown in Figure 1, which presents a structural rather than procedural description of the functional architecture.

The *TaskCommunicator* function of a DELS controller receives and issues task requests and responses to task requests, interacting with external DELS and with owned resources. For example, the DELS controller receives a task request, determines it can meet the request, responds to the requestor, then determines how an accepted task can be executed and issues task requests to one or more of its owned resources. Each received task request or response corresponds to some kind of event. The *EventDirector* function determines what kind of event has occurred and if a controller action is required it initiates that action. One class of actions is performed by the *ModelMaintainer* which maintains the necessary resource and task state information for the DELS. Another class of actions is decision-making in which the *AnalysisFormer* function uses the *ModelQuerier* function to obtain appropriate information from the *PlantTaskModel* in order to formulate an appropriate decision problem which is analyzed by the *Solver* function. If the result is the identification of a task that can executed, the *TaskDefiner* function translates the solver solution into an executable task which is communicated to the appropriate resource by the *TaskCommunicator* function. The realization of these controller functions will clearly depend upon the particular application.

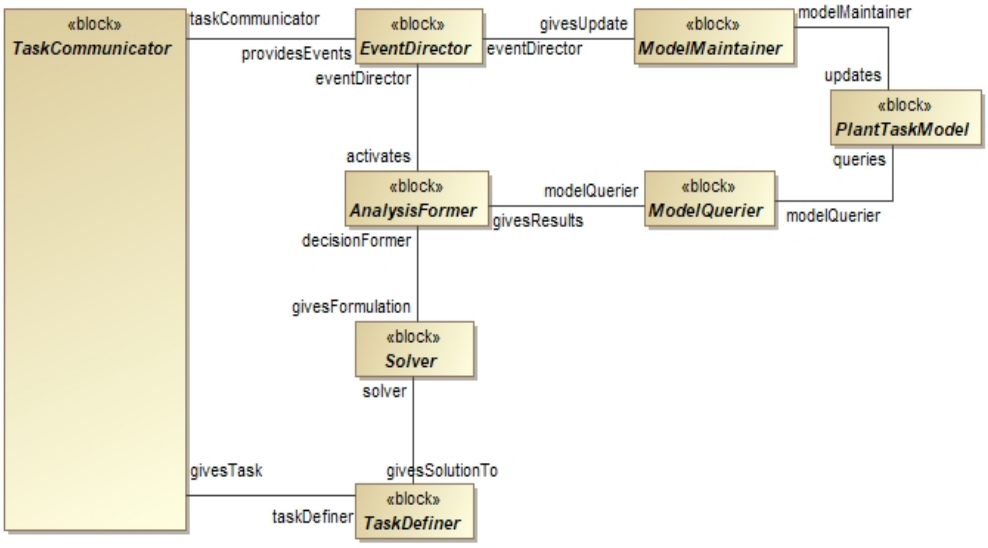

Figure 1 Controller functional architecture.

3 A USE CASE

Montreuil et al. (2021) describe an innovative approach to parcel logistics in which parcels at an originating hub are grouped into containers (in this case totes) based on their destination hub and are transported between hubs in standardized racks designed to fit within a standard truck or semi-trailer. A rack departing an originating hub may have totes for several different destinations, and travel through a network of intermediate hubs where the key function of the intermediate hub is to consolidate totes from arriving racks into departing racks for transport to their next destination. The hubs are DELS with the key resource type being a robotic consolidation cell (RCC) as illustrated conceptually in Figure 2.

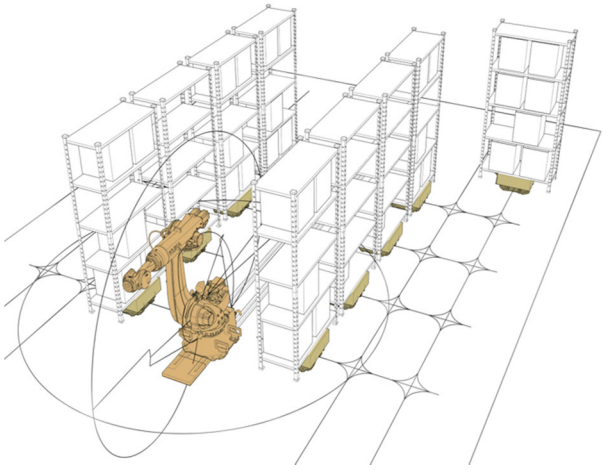

Figure 2: Conceptual robotic consolidation cell.

A hub might have many instances of the RCC in order to meet the peak throughput requirements. While control of the hub itself is a significant design challenge, the focus here is on the control within the individual RCC, which also is a DELS. The RCC will receive racks with manifests (identity of contained

totes and the tote destinations), and designation of racks as "strip" racks (racks with no assigned destination) or "stack" racks (racks with assigned destination). Individual totes may be moved from one RCC to another by a *ToteBot*. Within the RCC the *ShuffleBot* moves totes from strip racks to stack racks, or between a *ToteBot* interface and a rack. A hub-level controller will determine when racks or totes should enter or leave an RCC. The RCC also will receive information from the hub controller that may be used to determine priorities for rack consolidation, perhaps based on the target departure time of stack racks.

In Figure 3 a conceptual model of an RCC is shown in the dashed line box, with a cell controller, here identified as *ShuffleCellAgent*, with access to all relevant information about the state of the cell, i.e., the racks in the cells and the manifest for each rack, as well as the destination code for each rack. The destination code will identify the rack as a strip rack or a stack rack with a specific destination. The rack manifest will specify for each location in the rack, the contained tote ID (if occupied) and information usable to prioritize tote consolidation actions, such as target departure time. The RCC controller determines what tote move to make next, considering all the possible tote moves that will improve rack consolidation and the corresponding robot controller (identified as *ShuffleBotAgent*) will manage the execution of the tote moves by invoking the *ShuffleBotActuator* movement capabilities. The *ShuffleBotActuator* can access two kinds of locations, those where racks are positioned and those where a *ToteBot* can deliver or retrieve a tote. When a strip rack is depleted or a stack rack can no longer be improved, the RCC agent will notify the hub-level agent, here identified as *ShuffleCenterAgent*, which may have the identified rack removed and replaced.

In Figure 3 the "agent" designation is used in the sense of Luck and d'Inverno (2001), i.e., "an object with a non-empty set of goals". In the case of the *ShuffleCellAgent*, the goal is to achieve as much consolidation as possible, using the *ShuffleBot* in the shortest possible time, and for the *ShuffleBotAgent* the goal will be to achieve a specified tote movement, using the *ShuffleBotActuator* capabilities, in the most efficient way. The implementation of these agents will depend on the precise statement of the goals, the physical configuration of the *ShuffleCell,* and the technology of the *ShuffleBotActuator* and will determine the degree of optimization achievable.

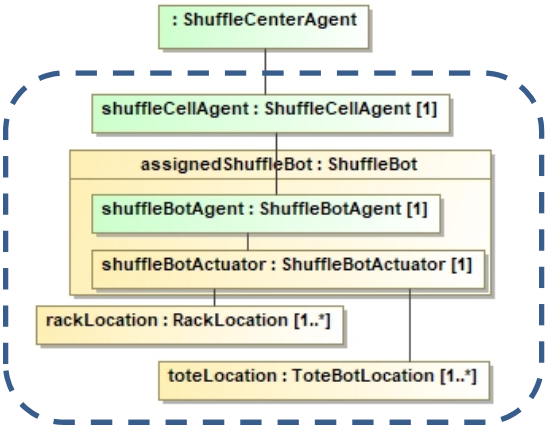

Figure 3: RCC physical and control architecture.

In summary, an RCC is a DELS with capabilities to consolidate totes from strip racks to stack racks or to move totes between rack locations and a designated transfer location where totes are placed for movement between RCCs. The set of racks in the RCC can be changed by the hub level controller, so the RCC also has the capability to update its rack set information; similarly for tote transfer information. Finally, it has the capability to notify the hub-level controller when it has exhausted all the consolidation-improving tote moves associated with a rack, or it has moved a tote from or to the tote transfer location.

For systems like the RCC, there are two major issues for simulation. One is to capture with sufficient fidelity the decision-making about the choice and sequencing of tote moves, i.e., the *logic* driving the system behavior, which is captured in the *ShuffleCellAgent* and, as we show, can be achieved in a repeatable manner by exploiting the modeling capabilities of AnyLogic™. The other is to model with sufficient fidelity the cycle time of the individual tote movements, i.e., the consequence of the control of the physical actuators that are part of the *ShuffleBotActuator*. A conventional agent-based or discrete-event simulation model either randomly samples a predefined distribution or uses tables of predefined constants for such cycle times. For that reason, we also have created an emulation of the RCC using Emulate3D™, which allows for physics-based emulation of robotic movements, from which cycle time distributions can be obtained for each kind of location-to-location movement by the robot. These empirically obtained cycle times then can be used in the AnyLogic™ simulation of the RCC.

4 RCC CONTROLLER CONCEPTUAL DESIGN

Controller design must specify how the functions identified in Figure 1 will be implemented. It is important to keep in mind that Figure 1 presents a functional architecture rather than an implementation architecture. In this section, we explain how this functional architecture can be realized from a conceptual perspective. Section 5 will describe a concrete instantiation.

Conceptually, *TaskCommunicator* function could be implemented as function calls between the three controllers. The hub-level controller (*ShuffleCenterAgent*) could call a function of the RCC controller to update the state of rack or tote positions when new racks or totes are added to the cell, or to instruct the RCC controller to place a tote in one of the tote transfer locations for removal. The RCC controller could call functions of the *ShuffleCenterAgent* to identify racks whose consolidation cannot be improved and to indicate that a tote has been removed from a tote transfer location. The RCC controller could call functions of the robot controller to transmit the next required tote move, and the robot controller could call a function of the RCC controller to indicate move completion.

EventDirector could be distributed in two ways. First, any call to a callable function represents an event, and the called function can make appropriate changes to the model data of the plant or task. Second, whenever a discrete event process completes (corresponding to some physical operation completion), the resulting state change could trigger an update the *PlantTaskModel*.

PlantTaskModel is first of all a collection of information showing the set of racks in the cell and the state of each rack with regard to its contained totes. In addition, the *PlantTaskModel* must maintain information that is needed to make decisions about tote movements, such as priorities, as well as information that may be used to compute performance metrics. This information may be local to the RCC or it could be maintained in some centralized database.

AnalysisFormer implements the logic for determining which, if any, tote move could be executed next. This involves determining the set of feasible moves (a tote that could be moved and an empty location to which it could be moved). Determining the set of feasible moves involves examining the rack and tote information in the *PlantTaskModel* and constructing the available move list. This function is triggered whenever the *PlantTaskModel* changes.

Solver requires selecting the best move from the list of potential moves and requires some metric for prioritizing the moves. For example, the best next move might be the one from the list of feasible moves involving a tote that has the earliest departure time. This function is triggered whenever the *AnalysisFormer* determines there is at least one feasible move.

TaskDefiner simply translates the selected tote move into the parameters needed to call the associated *ShuffleBotAgent* function.

We assume that each of the identified functions will have a well-defined signature of the form *functionName(set of input parameters, set of response parameters)*. The identification of the input and response parameters is particular to the technologies and the details of implementation.

The translation from concept to implementation must consider the target implementation platform and its capabilities for implementing these concepts.

5 RCC CONTROLLER IMPLEMENTATION IN ANYLOGIC™

Realizing the RCC controller conceptual design requires implementing the specified functions in a specific platform. In our AnyLogic™ simulation model of the RCC, all objects, including the racks, totes, tote movers, *ShuffleCenterAgent*, *ShuffleCellAgent* and *ShuffleBotAgent* are represented as AnyLogic™ agents. The agent representation enables information to be associated with objects, using both parameters and variables. For example, an AnyLogic™ agent representing a tote may have attribute values indicating its destination and its target departure time. An agent representing a tote mover may have parameters indicating its travel speed and time to dock at a transfer location. Using agents to represent the units of flow, the resources, and the controllers allows grouping agents into useful collections, e.g., a collection of the totes in a rack or a collection of the racks in a cell. The contents of collections are easily manipulated for selecting the "best" member of the collection.

Even though they represent only information, potential tote move tasks also are modeled as AnyLogic™ agents. This allows sets of potential moves to be manipulated as collections. For example, if all the feasible tote moves are in a collection, it is easy to find the one with the highest computed priority index.

AnyLogic™ agents also may have *functions*, for example a function that uses variables or parameters to compute attributes of the agent or to manipulate collections to which the agent belongs. For example, the agent representing a potential tote move may have a function to compute the priority for that tote move. The agent representing the *ShuffleCellAgent* may have functions that find all the (feasible) potential moves (using the state information for totes and racks) and that estimate their process times (using the location data for any identified move and performance information about the *ShuffleBotActuator*).

The *behavior* of the RCC is modeled in two ways. First, the logical behavior of the RCC controller is modeled using *state charts*. The AnyLogic™ state chart allows specifying *actions* to be taken upon entering or leaving a state, and it allows *conditions* under which transitions between states can occur. Both the actions and the conditions can involve significant computation and are coded in Java™. State charts are very convenient for modeling the logical behavior of the RCC controller.

The operational behavior of the RCC resources, for example the robot actuator, is modeled using AnyLogic™ blocks *enter*, *exit*, *delay*, *moveTo*, and *service*. When a task is assigned to a resource, the corresponding agent joins an empty, single-unit capacity queue, immediately moves to the assigned resource and is delayed for the duration of the operation, then moves to an empty single-unit capacity queue and leaves.

This approach to implementing the RCC controller allows clear separation of plant and control, as illustrated in Figure 4. The state chart at the left of the figure captures part of the logic of the RCC controller, specifically the states in which the controller may be found and the transitions between states.

Once totes and racks begin arriving to the RCC, it is in the *ReadyToShuffle* state and it transitions to *Shuffling1* state by executing the code shown for *transition20*, which determines the best tote move to make (if there is a feasible move) and orders it to be executed by calling the *f_orderShuffle()* function. The *f_orderShuffle* function takes the selected potential move, *moveToDo*, and sends it to the process at the lower left corner of the figure. When that process is complete, the statechart transitions from *Shuffling1* to *delay4* where the state is updated to reflect the tote move and the process repeats.

The pattern of implementation shown in Figure 4 is repeated for the processes of adding racks or totes to the RCC or removing them. The implementation for these processes creates events associated with the completion of a process, and these events are handled in the *addTBandRack1* state where appropriate changes are made to the *PlantStateModel*, i.e., the attribute values of the relevant agents.

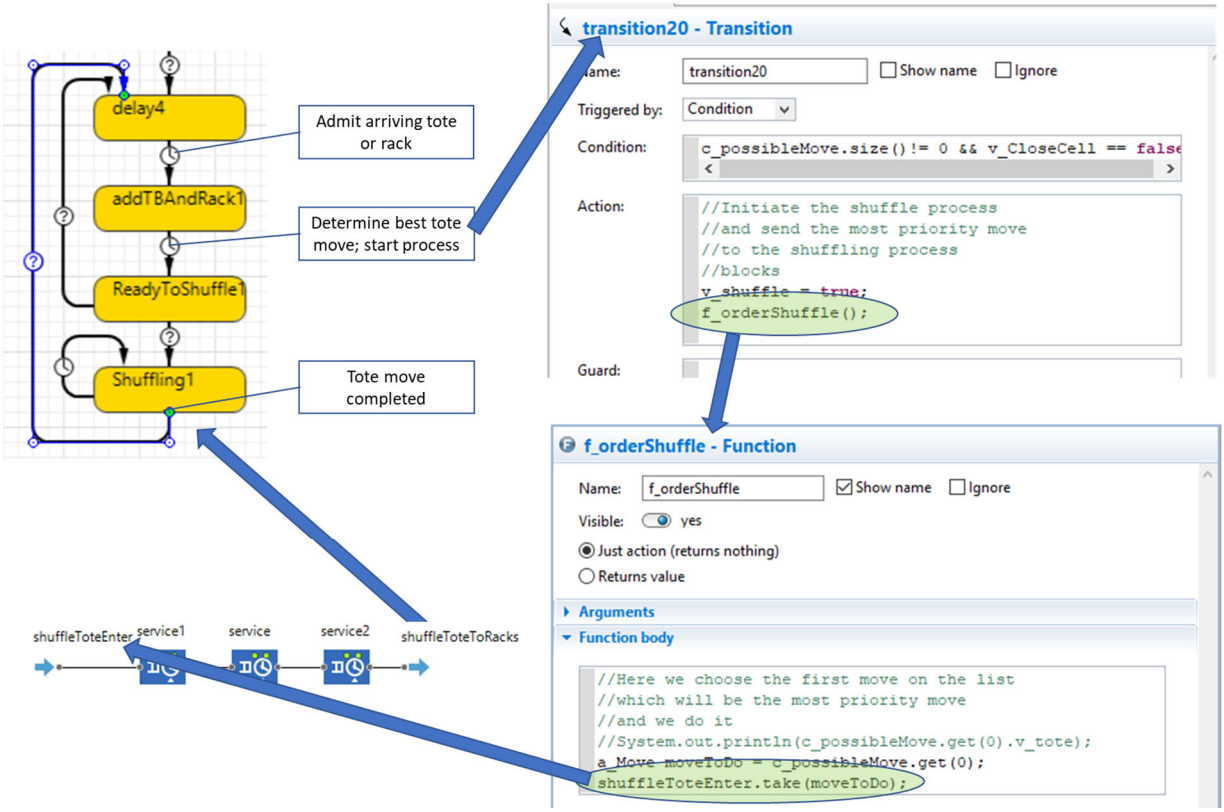

Figure 4: Separating plant and control in AnyLogic™.

Returning to Figure 1, the *PlantStateModel* is captured in the collections of agents with their parameters and attribute values. The events corresponding to controller state changes cause functions to be called that update the state model. At any point in time, the state model gives a complete account of the rack and tote transfer positions in the RCC as well as the specific racks and their constituent totes. If one felt it necessary, a collection could gather all these agents into one "state collection". The *PlantStateModel* function is implemented in this manner.

The *AnalysisFormer* function from Figure 1 is implemented in the state *Shuffling1* in Figure 4 when the collection of potential moves is updated.

The *Solver* function from Figure 1 also is implemented in the state *Shuffling1* in Figure 4 simply by sorting the collection of potential moves by decreasing urgency.

The *TaskDefiner* function from Figure 1 also is implemented in *Shuffling1* in Figure 4 by passing the selected move, modeled as an agent, to the process model for execution.

All the controller functions identified in Figure 1 are implemented in the combination of state charts and functions. Of course, the RCC is a very simple DELS, with only one externally observable capability, i.e., to move totes between racks and tote transfer locations in order to improve the consolidation of selected racks. While the intent here is not to discuss the entire hub simulation, it is worth noting that the plant-control separation illustrated in Figure 4 is replicated at the next higher level of the hub simulation model, where the control decisions include when to initialize or shut down an RCC, which racks to send to an RCC, what to do with racks that are removed from an RCC, and when to move totes between two RCCs.

The specific implementation of the *AnalysisFormer* and *Solver* functions may be based on a very simple heuristic (e.g., choose the task corresponding to the tote with the least remaining time to scheduled departure), or it could be based on a complex optimization (e.g., sequencing the feasible tote moves to minimize the unloaded robot travel). In fact, the implementation approach described here can support

multiple *AnalysisFormer* and *Solver* implementations with a simple switch function to select the one to be used at any point in the simulation.

6 RCC IMPLEMENTATION IN EMULATE3D™

AnyLogic™ proved to be a very capable platform for developing the hybrid agent-discrete event simulation, but it has one significant drawback in terms of evaluating novel technological implementations. The process times themselves cannot be estimated from the AnyLogic™ model but must be provided from some external source. Without a physical laboratory in which to experiment, this presents some challenges.

There are several software products available today that allow physics-based modeling of devices and systems. These kinds of models tend to be quite time-consuming to develop and are not easily modified to reflect alternative arrangements or technology selections. The benefit of using them would be the resulting estimates of process times that could then be transferred to more conventional simulation models where alternative configurations are more easily modeled and tested.

We chose to use Emulate3D™ to develop a physics-based model of the RCC. A rendering of the initial model is shown in Figure 5. This model was used to develop point-to-point move times for the robot, both with and without a tote.

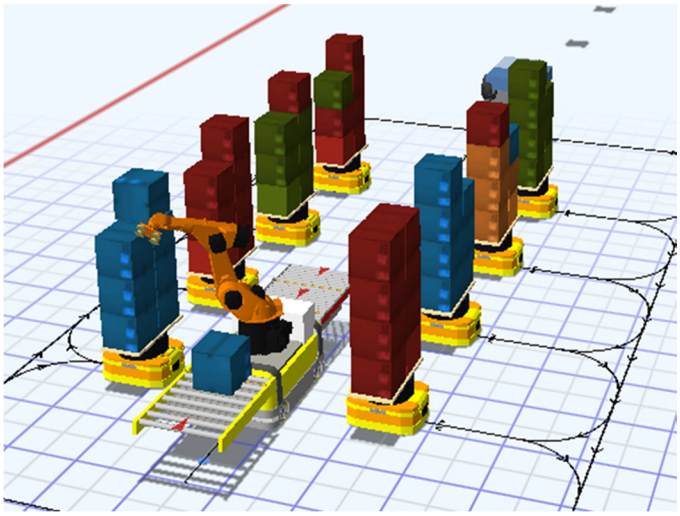

Figure 5: VR rendering of RCC.

Ideally, exactly the same RCC controller implementation would be possible for both the AnyLogic™ and Emulate3D™ models. However, differences between the two platforms eliminate the possibility of directly porting the AnyLogic™ controller implementation into the Emulate3D™ model.

Fortunately, at this time, our primary objective in creating the emulation is to develop plausible cycle time distributions for the point-to-point movements of the *ShuffleBot*, both with and without a tote. These cycle time distributions are not likely to be dependent upon the sequencing of the moves, so even if the two controller implementations do not produce exactly the same sequence of moves, the emulation results still will be appropriate for use in the hybrid agent/discrete-event simulation.

This approach is not without significant challenges. Creating a physics-based simulation of technology that does not yet exist requires making assumptions and designing controllers that direct the sensors and actuators of the modeled physical system. Significant development may be needed to optimize manipulator trajectories in order to minimize operation cycle times, and the estimated cycle times should be viewed as conservative and subject to improvement.

7 INITIAL EMPIRICAL RESULTS

The Emulate3D™ model was used to develop cycle time distributions for possible point-to-point *ShuffleBot* moves. Our implementation of the *ShuffleBot* combined a transport platform and a conventional robot. To create a cycle time model, we used Emulate3D™ to estimate the translation times of the transport platform, and the times for the robot to move from one possible tote location to another, both with and without a tote. Tote locations are designated by a level (1-4) and a side (right or left). Table 1 contains a sample of the kinds of results that can be developed using Emulate3D™. Cycle times can be computed by combining these two operation times so the AnyLogic™ model does not have an explicit representation of the *ShuffleBotAgent*.

While observing the operation of a single *ShuffleCell* is important to verify the *ShuffleCell* is behaving as planned and expected, it is not adequate for assessing the *ShuffleCell* concept as the basis for proposing an innovative parcel logistics hub. To do that, a full hub simulation is required. As described by Montreuil et al. (2021), this hub has receiving and shipping functions based on racks containing totes; staging areas for racks after receiving or prior to shipping and a *ShuffleCenter* containing instances of the *ShuffleCell* as well as instances of a companion *BufferCell* where racks may be stored temporarily when they are not assigned to a specific *ShuffleCell*. The *ShuffleCell* and *BufferCell* have the same footprint, allowing for considerable flexibility in the physical arrangement of functions within the *ShuffleCenter*. An example of one potential realization of the full hub is shown in Figure 6, which is a snapshot from an executing AnyLogic™ full hub simulation.

Table 1: Sample of Emulate3D(tm) results.

Transport Distance	Time		Robot Movement	Loaded	Empty
1	4.666291		cross aisle translate left	9.501862	2.342238
2	6.418823		cross aisle no translation	6.052915	3.584093
3	8.389416		cross aisle, translate left, move up 1 level	12.51716	4.139916
4	10.21675		cross aisle, no translation, move up 1 level	7.126905	3.484807

The initial experimentation uses parcel shipment data from a conventional hub, where approximately 400,000 parcels per day were received, sorted and shipped. The parcel data were analyzed and then parcels were containerized into totes, i.e., a single tote would contain a set of parcels that would fit in the tote, be coming from a single origin and have the same destination. For the synthetically containerized totes, the assigned arrival time corresponded to the original arrival time of the contained parcels, and the assigned target departure time corresponded to the original departure time of the contained parcels.

Given this input data set and the configuration shown in Figure 6, which has 30 *ShuffleCells* and five *BufferCells*, the simulated operation met all departure time targets. The simulation run time is approximately six minutes on an EPYC Processor at 2,500Mhz, 190 Gb of physical memory.

One very simple preliminary experiment was conducted by varying the number of stack versus strip racks assigned to cells, with the assignment varying from (7 stack, 1 strip) to (1 stack, 7 strip). This design parameter is likely to have some important and subtle impacts on overall hub performance, and one measure of that would be the time required to complete all the shipments. Results for this experiment are displayed in Table 2, where finish time is in simulated seconds.

Clearly a great deal of further development and experimentation is required to fully explore the design and implementation of the RCC controller and the other DELS controllers contained in this new concept for a parcel logistics hub. All of that future work will follow the pattern that has been described and illustrated here, i.e., careful separation of plant and control models, using agents to represent every object in the simulation, implementing the controllers using state machines with Java-based actions and conditions, and using the *enter*, *exit*, *delay*, *moveTo*, and *service* blocks to represent process durations. In fact, this approach already has been implemented in the full hub simulation.

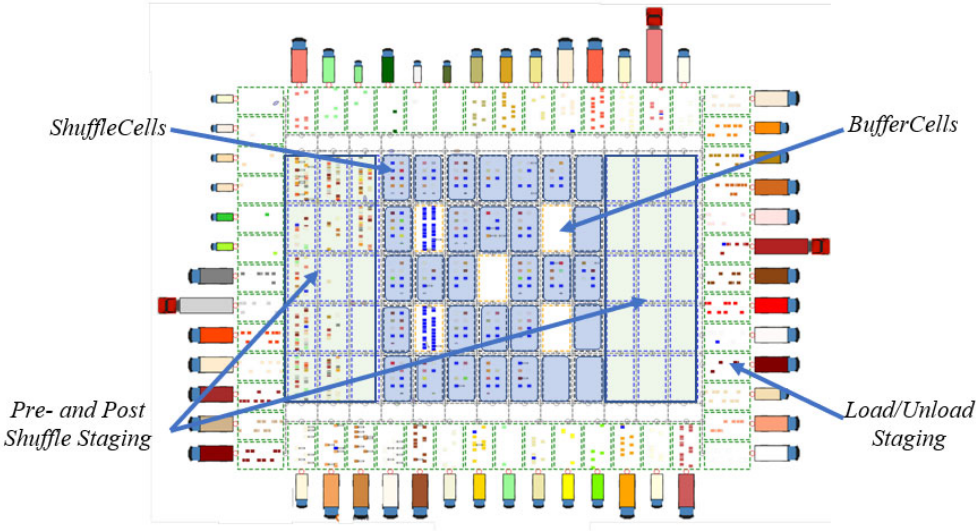

Figure 6: Snapshot of full hub simulation (Montreuil et al. 2021).

Table 2: Varying an operational control parameter and resulting finish time in simulated seconds.

ShuffleCell configuration		
# Stack Racks	#Strip Racks	Finish Time
7	1	85000
6	2	87000
5	3	87000
4	4	87500
3	5	87600
2	6	86000
1	7	91000

8 OBSERVATIONS AND IMPLICATIONS FOR FUTURE RESEARCH

Simulations are important for designing both the physical configurations and control systems of production and logistics systems. Translating the configurations as simulated into configurations as implemented can be quite challenging if the simulation model does not carefully separate control models from plant models. This paper illustrates one approach to achieving that separation when using the AnyLogic™ simulation platform.

It is not unreasonable to view the state charts with their associated code for entry and exit actions and transition conditions as a "design specification" for the corresponding operational controller implementations. Simulations where there is not a clear separation of plant and control are much less useful as control system design specifications or requirements.

The operational controller functional requirements identified in Sprock and McGinnis (2015) and the functional architecture proposed in McGinnis (2019) have been used in the simulation implementation of operational controllers for the described use case. These kinds of logistics hubs may seem operationally relatively simple compared to more complex DELS like aircraft assembly, semiconductor fabrication, or hospitals, so additional research is needed to establish the general applicability of the demonstrated approach.

The plant-control separation as implemented for this specific use case is somewhat *ad hoc*. It is reasonable to ask if it is possible to abstract this use case and the demonstrated modeling approach to create a

general purpose methodology for simulation modeling of DELS, using AnyLogic™. Then, of course, it would be reasonable to investigate how such methodology could be adapted to other simulation platforms.

One of the most appealing promises of "smart factories", Industry 4.0, and "digital twins" is the ability to maintain a digital representation of the target system, update its state to reflect the actual state of the target system, and run "what if" analyses of alternative planning and operational control decisions using the "digital twin" to predict results in the physical system. For systems with "smart" controllers, this promise simply cannot be met if those simulations cannot correctly reflect the operational control decisions with a high degree of fidelity. To do so, the logic for making those operational control decisions that is coded into those "digital twins" must precisely reflect the logic of the operational controller in the physical system. In the past, neither the operations research community nor the simulation community has addressed this issue. This paper offers at least one approach to creating the promised "digital twin". This may not be the only way, or the best way, but it does provide a starting point for an extremely important line of research and development.

ACKNOWLEDGEMENTS

The work reported here has been supported by research grants from NIST (70NANB15H234) and projects with industry partners, including Boeing and SFX. Mr. Hugo Hamon did excellent work in developing an initial implementation of the robotic consolidation cell.

REFERENCES

Bueno, A., M. G. Filho, and A. G. Frank. 2020. "Smart Production Planning and Control in the Industry 4.0 Context: A Systematic Literature Review". *Computers & Industrial Engineering* (149): 106774.

International Society of Automation. 2021. https://isa.org. Accessed 2 June.

McGinnis, L. F. 2010. "The Future of Modeling in Material Handling Systems". In *11th International Material Handling Research Colloquium,* June 21-24, Milwaukee, WI, 261-274.

McGinnis, L. F. 2019. *Formalizing ISA-95 Level 3 Control with Smart Manufacturing System Models.* NIST Interagency/Internal Report. https://doi.org/10.6028/NIST.GCR.19-022, accessed 2nd June.

Montreuil, B., L. McGinnis, S. Buckley, S. Babalou, W. Bao, and A. Beranji. 2021 (to appear). "Physical Internet Induced Parcel Logistics Hub Innovation". *International Physical Internet Conference*, June 14-16.

Rockwell Automation. 2021. https://www.demo3d.com. Accessed 2 June.

Sprock, T. 2016. "A Metamodel of Operational Control for Discrete Event Logistics Systems". https://smartech.gatech.edu/handle/1853/54946, accessed 2 June.

Sprock, T., C. Bock, and L. F. McGinnis. 2019. "Survey and Classification of Operational Control Problems in Discrete Event Logistics Systems (DELS)". *International Journal of Production Research*: 5215-5238.

Sprock, T., and L. F. McGinnis. 2015. "A Conceptual Model for Operational Control in Smart Manufacturing Systems". *15th IFAC Symposium on Information Control Problems in Manufacturing.* IFAC-PapersOnline:1865-1869.

Sprock, T., G. Thiers, L. McGinnis, and C. Bock. 2019. *Theory of Discrete Event Logistics Systems (DELS) Specification.* NIST Interagency/Internal Report. https://nvlpubs.nist.gov/nistpubs/ir/2020/NIST.IR.8262.pdf, accessed 2 June.

The AnyLogic Company. 2021. https://the.anylogic.company/. Accessed 2 June.

AUTHOR BIOGRAPHIES

LEON McGINNIS is Professor Emeritus in the Stewart School of Industrial and Systems Engineering at The Georgia Institute of Technology, where he has held leadership positions in faculty governance, academic programs, and research centers. His research is focused on developing the methods and tools necessary for systems engineering of discrete event logistics systems, including design, planning, management, and control issues. His email address is leon.mcginnis@gatech.edu and his work is featured on the website http://factory.isye.gatech.edu.

SHANNON BUCKLEY is a Ph.D. candidate in the H. Milton Stewart School of Industrial and Systems Engineering at The Georgia Institute of Technology. His research interests include warehouse systems and processes design, where he focuses on creating innovative solutions to current problems faced in the parcel logistics industry. He then validates these solutions with complex, high-fidelity simulations in the AnyLogic simulation modelling language. His email address is sbuckley8@gatech.edu.

ALI BARENJI is a Sr research scientist in the H. Milton Stewart School of Industrial and Systems Engineering, at the Georgia Institute of Technology. Before joining Georgia Tech, he has served as a lecturer in the Mechatronic Engineering Department at Kennesaw State University. He was also a visiting researcher at Massachusetts Institute of Technology and a postdoc fellow in the Guangdong University of Technology. His research is focused on smart manufacturing and developing the methods and tools necessary for systems engineering of discrete event and multi-agent logistics systems. His email address is abarenji3@gatech.edu.