NIST GCR 19-022

Formalizing ISA-95 Level 3 Control with Smart Manufacturing System Models

Leon F. McGinnis

This publication is available free of charge from:
https://doi.org/10.6028/NIST.GCR.19-022

National Institute of
Standards and Technology
U.S. Department of Commerce

NIST GCR 19-022

Formalizing ISA-95 Level 3 Control with Smart Manufacturing System Models

Prepared for
U.S. Department of Commerce
Engineering Laboratory - Systems Integration Division
National Institute of Standards and Technology
Gaithersburg, MD 20899

By
Leon F. McGinnis
The Georgia Institute of Technology,
School of Industrial and Systems Engineering

This publication is available free of charge from:
https://doi.org/10.6028/NIST.GCR.19-022

November 2019

U.S. Department of Commerce
Wilbur L. Ross, Jr., Secretary

National Institute of Standards and Technology
Walter Copan, NIST Director and Undersecretary of Commerce for Standards and Technology

Disclaimer

This publication was produced as part of contract 70NANB15H234 with the National Institute of Standards and Technology. The contents of this publication do not necessarily reflect the views or policies of the National Institute of Standards and Technology or the US Government.

Acknowledgements

The work reported here would not have been possible without the financial support of the National Institute for Standards and Technology, and particularly the mentoring of Conrad Bock. At the same time, it would not have been possible without the participation, encouragement and support of many others. Jack Harris of Rockwell Collins (now Collins Aerospace) provided the first financial support of work that used SysML to model a production system and automated the creation of Arena simulation models. Sandy Friedenthal, then at Lockheed, sponsored an early multidisciplinary project to integrate product system and production system design. Boeing has been a strong supporter, through the Boeing-Georgia Tech Strategic University Partnership, for production system modeling with SysML and simulation model automation; Michael Christian and Adam Graunke have been key collaborators. McKesson High Value Solutions provided both financial support and very important access to actual central fill pharmacies. Dr. George Thiers, founder of MBSE Tools, Inc., and Dr. Tim Sprock, currently at NIST, have made seminal contributions to these ideas and both have been directly involved, at one time, in the work reported here. Many graduate students in the Steward School of Industrial and Systems Engineering at Georgia Tech have participated in this project and earlier work on which this project was based, most recently including Morgan McCombs, Chinmay Navrange and Prabodh Gawande. Any errors of omission or commission in this document are the sole responsibility of the author.

Abstract

Achieving the vision of ”smart manufacturing and ”Industrie 4.0” requires building on successes in computational control of processes to create generic approaches for management of manufacturing operations, or smart manufacturing operations management (SMOM). The approach to SMOM presented in this report generalizes the modeling framework of ISA-95 with a reference model for discrete event logistics systems (DELS) that identifies five generic operations management decisions. This report applies it in a computational model of a large-scale, highly-automated central fill pharmacy that is the basis for a simulation testbed enabling convenient experimentation with operations management policies and decision algorithms.

Key words

Smart Manufacturing; Operations Management; Systems Modeling.

TABLE OF CONTENTS

Table of Figures

EXECUTIVE SUMMARY

This report demonstrates a novel approach to modeling production systems, including their operational control, with a large-scale, highly automated central fill pharmacy as the demonstration use case. The products are: (1) the system model, authored using the OMG SysML™ language; and (2) a discrete event simulation of the modeled central fill pharmacy, structured specifically to support convenient experimentation with alternative operations management policies and decision algorithms.

The approach is based on a semantic reference model for discrete-event logistics systems (DELS) that enables modelers to create a comprehensive and computational model of structure, behavior and control of DELS. This model is described in Chapter 2 and shown to generalize the resource and process models underlying the ISA-95 standard. A key element of the reference model is its represents operational control. Chapter 3 discusses generic operational controller requirements, functions, and a suggested architecture.

The demonstration use case is automated central fill pharmacies (CFPs). These are described in Chapter 4, which focuses on a particular style of CFP, and is based on a particular CFP. However, no proprietary information is presented.

Chapter 5 applies the DELS reference model to establish some abstract components of a CFP model. This chapter bridges the reference model of chapter 2, the specific use case of Chapter 4, and a detailed system model, presented in Chapter 6.

The detailed model of a particular CFP is presented in Chapter 6. This model is the first such model to be available in the public domain, as far as we know, because OMG SysML™ has almost exclusively been applied to models of product systems, such as airplanes and space missions. The extant papers that do address production or manufacturing present only very limited models of small systems. A major challenge addressed in Chapter 6 is how to organize large production system models. Other challenges addressed include representation of: order filling processes when both the number of specific drugs ordered and their identities are not known *a priori*; the mechanisms by which operational controllers invoke the behaviors of resources; and material transport and process contingencies in the operational controller.

Chapter 7 describes a simulation model reflecting the structure, behavior and control captured in the system model of Chapter 6. Most commercial off-the-shelf discrete-event simulation packages have very limited native capability for controller decision making, requiring the modeler to be able to code such decision making in the underlying language and link this new code with the existing package. A key feature of the model in Chapter 7 is support for integrating a generic simulation platform providing all the standard discrete event simulation capabilities with a generic mathematical analysis tool in which key elements of the controllers are implemented. Some initial results from experiments with the simulation model are discussed.

Much has been accomplished and much has been learned in this project. However, there remain many issues to resolve before the technology for SMOM achieves a commercially viable technical readiness level. Chapter 8 summarizes the lessons learned and identifies key areas for further research and development.

1 Introduction, Motivation and Approach

1.1 Introduction

"Smart manufacturing" represents the next major phase in the evolution of manufacturing and aims to make the capture, dissemination, and intelligent use of information for decision-making reliable, fast, cheap, and ubiquitous. When fully realized, smart manufacturing technologies and methods will impact every aspect of manufacturing. Both quality and time to market will be improved—integrating product design and product manufacturing allows product information to flow seamlessly to production and producibility information to flow seamlessly back to design. Costs will be reduced—smart operations management improves production resource utilization and responds faster to contingencies through sensing and real-time decision making.

The smart manufacturing transformation extends from the decisions on the shop floor about how to sequence tasks or respond to contingencies all the way to the executive suite where decisions are made about products, markets and supply chains. Achieving smart manufacturing across this broad spectrum of decision-making requires:

- Data/information interoperability: The executive suite deals with time intervals of months or years, addresses families of products, geographical regions, and transactions with suppliers and customers. Decisions on the shop floor deal with seconds or minutes, with specific machines, workstations and people, and the individual operations required to transform material into product. Across the enterprise, decision making requires information aggregation/disaggregation, which is possible only with high-quality detailed reference models for the information being exchanged.
- Decision-support analysis automation: At all levels of the manufacturing enterprise, the fundamental decisions are "what to do," "how to do it" and "when to do it". Making these decisions requires an understanding of the alternative actions that can be taken regarding products, the processes for their production, and the resources available to execute those processes, and the decisions themselves must be made considering the impacts on time, quality and cost. Automatically formulating and solving appropriate decision-support analyses is possible only with high-quality detailed reference models for the products, processes and resources.

The research reported here addresses the second of these fundamental requirements in the context of attempting to "enable real-time monitoring, control, and performance optimization of smart manufacturing" (http://www.nist.gov/el/ smartcyber.cfm). Our approach is to use a single modeling language (OMG SysML™) to construct a standard representation of the manufacturing system that explicitly formalizes plant and control separation for the domain. The resulting architecture for smart manufacturing provides the missing bridge between system models and data and analysis models and methods in order to enable operations management decision support. While there are existing models that capture the structure and behavior of the manufacturing plant, there remains a need for an explicit model of operational control that can bridge between system models, analysis models, and execution tools.

This approach is consistent with and draws insight from the "model-based systems engineering" (MBSE) approach that is rapidly becoming standard practice in the aerospace and defense industry, for the development of "product" systems such as airplanes and weapons systems. Until now, MBSE has not be developed or deployed for the design of the factories or supply chains producing these systems.

1.2 What Is the Opportunity?

Figure 1-1 illustrates the manufacturing context for control as defined by ISA-95. There is a "base system" consisting of physical resources that convert material flows from an input state to an output state. ISA-95 addresses the decision making and execution control functions that determine the goals for the base system and make those goals executable. To do this, ISA-95 separates the manufacturing domain into four levels: 4) Business Planning & Logistics (the domain of ERP systems), 3) Manufacturing Operations Management (the domain of MES) 2) Manufacturing Control Systems (the domain of PLC, DCS, SCADA), and 1) Intelligent devices. The primary focus of ISA-95 is on defining the *functions* required for achieving control and the *information* that must be exchanged between these functions.

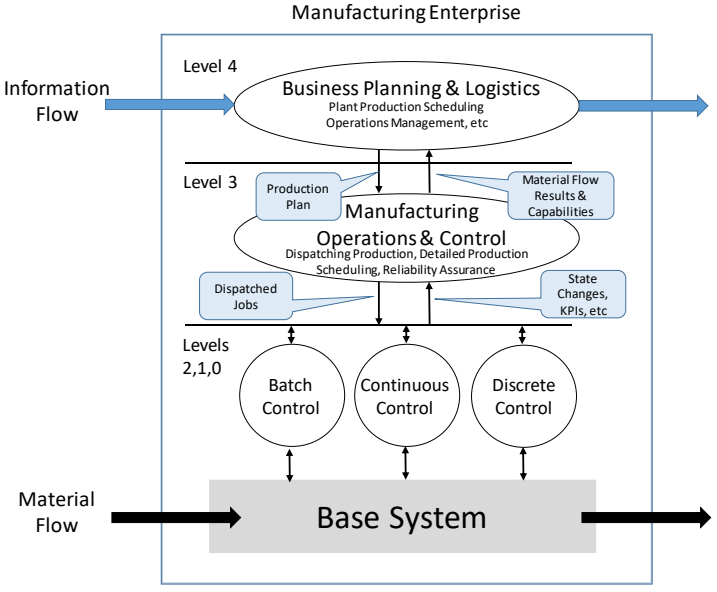

Figure 1-1 Manufacturing Enterprise Context for ISA-95

At ISA-95 Levels 0,1 and 2, control consists of executing predefined operations, such as running a part program on a CNC mill, or assembling a set of parts. In this domain there is a large and robust research and development literature on implementing control from the computer science, mechanical, electrical, and software engineering perspectives. With the evolution towards advanced manufacturing systems (AMS), much has been accomplished in designing and managing these complex systems, including research in topics such as: how is the control network organized (Dilts, Boyd, & Whorms, 1991), how should control networks be implemented (Galloway & Hancke, 2013), how can legal sequences of controller actions be generated (Smith, Joshi, & Qiu, 2003)), how to generate PLC code (Vogel-Heuser, Witsch, & Katzke, 2005), and how to use automata and formal language theory to derive the existence and structure of controllers, and how to define a controllable language for discrete event (Davis, Jones, & Saleh, 1992). While there is always room for advancement, this level of control appears to be reasonably well-understood, with reasonably effective and robust solutions available in manufacturing.

ISA-95 level 4 is concerned with developing a production plan, i.e., determining what products should be produced and when they should be available, where the time intervals of concern can be week, months, or even quarters. Within the operations research and management science disciplines, there is a vast literature on *production planning*, which tends to be focused primarily on formulating and solving large scale mathematical optimization problems in order to find the "best" production plan—expressed as quantities of product delivered by time period, subject to estimates of demand, production capacities, and costs of production, warehousing and transportation, see, e.g., (Jans & Degraeve, 2008). In this literature, the "problem is solved" when a vector of decision variable values is obtained. While research in this topic continues apace, there also are a number of commercial software solutions that provide decision support. In the ISA-95 framework, the resulting production plan is passed from Level 4 to Level 3.

ISA-95 Level 3 consumes the production plan, creates a production schedule, consisting of individual processing steps, perhaps with start/finish times, and invokes the execution capabilities of Levels 2, 1 and 0. Again, within operations research and management science, there is a vast literature on *production scheduling*, which assumes there is a set (or perhaps a stream) of job types with known resource requirements, that need to be assigned to available production resources in some sequence, see, e.g., (Silver & Peterson, 1998). This is well known to be, in most cases and from a theoretical perspective, an extremely difficult decision problem (most likely admitting no exact computational method which can reach a guaranteed optimum decision in time that is less than exponentially related to the problem size). In this literature, the "problem is solved" when the dispatching or scheduling decisions have been determined, again, typically as a vector of release or start times. Ultimately, Level 3 must translate the production schedule into tasks or jobs that are executable by the base system, through the control actions at Levels 2, 1 and 0.

There is a draft proposal within ISA-95 for the control functions at level 3, as summarized in Figure 1-2 below. There are eight identified functions:

- Detailed scheduling: translates the operations schedule from Level 4 into work schedules
- Resource management: responsible for defining, acquiring, adjusting and retiring resources
- Dispatching: assigns work scheduled to resources
- Tracking: uses data collected from Level 1 and 2 to update schedule progress and provide input to dispatching and performance analysis
- Definition management: defines the work process
- Execution management: interacts with Level 1 and 2 functions to execute the defined and dispatched work
- Data collection: aggregates data on work commenced, redirected, completed, aborted, executed or reconciled

In this framework, there are three decision functions: scheduling, resource management, and dispatching.

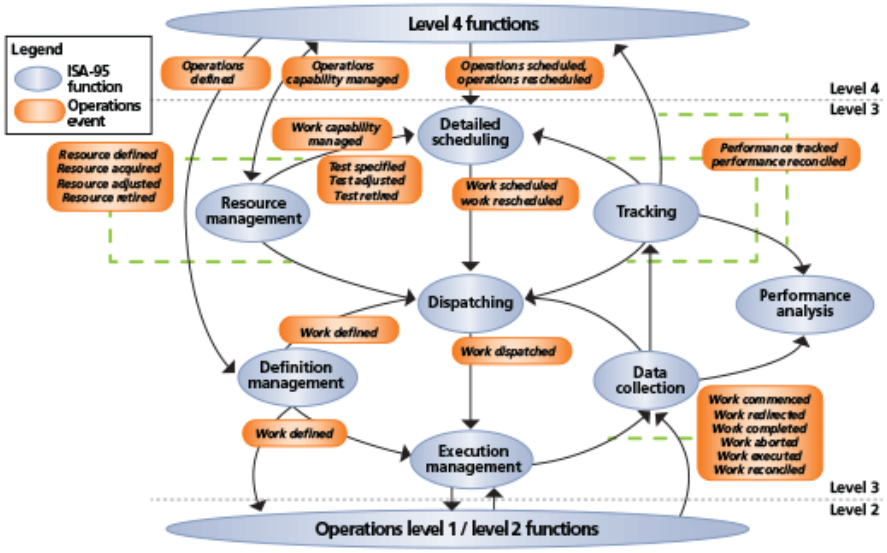

Figure 1-2 Functions and messages in Level 3

Source: https://www.isa.org/intech/20171203/

Deploying this framework in practice will require concrete implementations of the functions themselves, and the processes for synchronizing their execution. The concrete implementation of a decision function, e.g., Dispatching, will require converting the information available to the function, Work scheduled, Work defined, Work dispatched, and the status of each individual Work dispatched, into a decision problem. The decision required is the work to dispatch next. Conceptually, solving this decision problem requires, either explicitly or implicitly, predicting the outcomes of alternative decisions. It requires an understanding, not just of the parts of the system, but of their interactions as well. In other words, it requires an understanding of the structure and behavior of the Level 2 controls and the base system.

While ISA-95 has defined the events and corresponding information exchanges, as well as the operational control functions, it does not provide a mechanism for specifying the base system or the control architecture. The opportunity addressed in this report is the development of these important aspects of ISA-95 deployment.

ISA-95 Level 3 events are *discrete events*. The information being exchanged represents a discrete message. Moreover, the work being dispatched represents a discrete unit of work, with a planned and observed start/finish time. Thus, the system represented in the ISA-95 Level 3 framework can be viewed as a *Discrete Event Logistics System* (DELS), defined in (Sprock, Thiers, McGinnis, & Bock, 2019) as:

- a network of resources, arranged in a facility; each resource has one or more processing capabilities, with a capacity for each capability;
- products flow through this network of resources, transformed by processes executed by the resources; a process may require capabilities of more than one resource; processes can change location, age, or condition of products.

The DELS reference model is developed explicitly to support specifying the parts, interfaces, and behaviors of resources, to enable automation of decision-support analyses.

1.3 Approach

The work reported here demonstrates that the DELS reference model supports deployment of the ISA-95 Level 3 reference model. The DELS reference model supports developing a base system model expressed in OMG SysML ™, including the specification of a control architecture conforming to ISA-95. In addition, a generic controller architecture for DELS is used to specify ISA-95 control function implementation.

The demonstration use case is a central fill pharmacy producing approximately 30,000 prescriptions per day using a combination of dispensing automation and manual dispensing.

The DELS model of the central fill pharmacy constitutes a design document for developing a corresponding discrete event simulation model implemented in MATLAB/SimEvents.

1.4 Overview

Chapter 2 discusses the relationship between the ISA-95 reference model and the DELS reference model.

Chapter 3 identifies operational controller requirements, functions and a suggested architecture.

Chapter 4 provides a detailed description of the test case, the central fill pharmacy.

Chapter 5 applies the DELS reference model to define some abstract components of a CFP model. This chapter bridges the reference model in Chapter 2, the specific use case in Chapter 4, and detailed system model, presented in Chapter 6.

Chapter 6 presents the detailed SysML model of the demonstration case, addressing resources, processes, and controls.

Chapter 7 describes a discrete event simulation of the modeled CFP and presents some initial results from experimentation using the simulation.

Chapter 8 discusses the results and conclusions from this work and identifies important avenues for further research and development.

2 ISA-95 and DELS

ISA-95 specifies a reference model for the information exchanged between the various functions required for manufacturing enterprise planning and control. This includes information about the product the resources available, the operations required to produce the product and schedules for production. The DELS reference model, developed expressly to support design and operational decision making, specifies product, process, resource, facility and control. Because the DELS reference model is used here, it is important to understand how the two reference models are related, and in particular, any differences in how they structure information about manufacturing operations management. The description of ISA-95 given here will necessarily be brief and incomplete but will include the essential aspects of ISA-95 that have corresponding elements in the DELS reference model.

2.1 DELS Overview

The basic DELS ontology is shown in Figure 2-1. *Product* is defined by a bill of materials identifying the individual parts and assemblies which also may be (intermediate) *Products*. A *Product* is created by a *Process*, an activity, which may employ other *Process*es. The *Process* may require inputs of *Resources*, which can include *Material* but also *ActiveResource*s, such as *Equipment* or other *DELS*. A *Process* is executed by a capable *ActiveResource* which can be a collection of resources, and that execution is authorized by a *Task*. Not explicit in Figure 2-1is an assumption that *Task* is issued by some controller performing the ISA-95 control functions and is specific to a particular *Process*. The complete DELS reference model can be found at (Sprock, Thiers, McGinnis, & Bock, 2019).

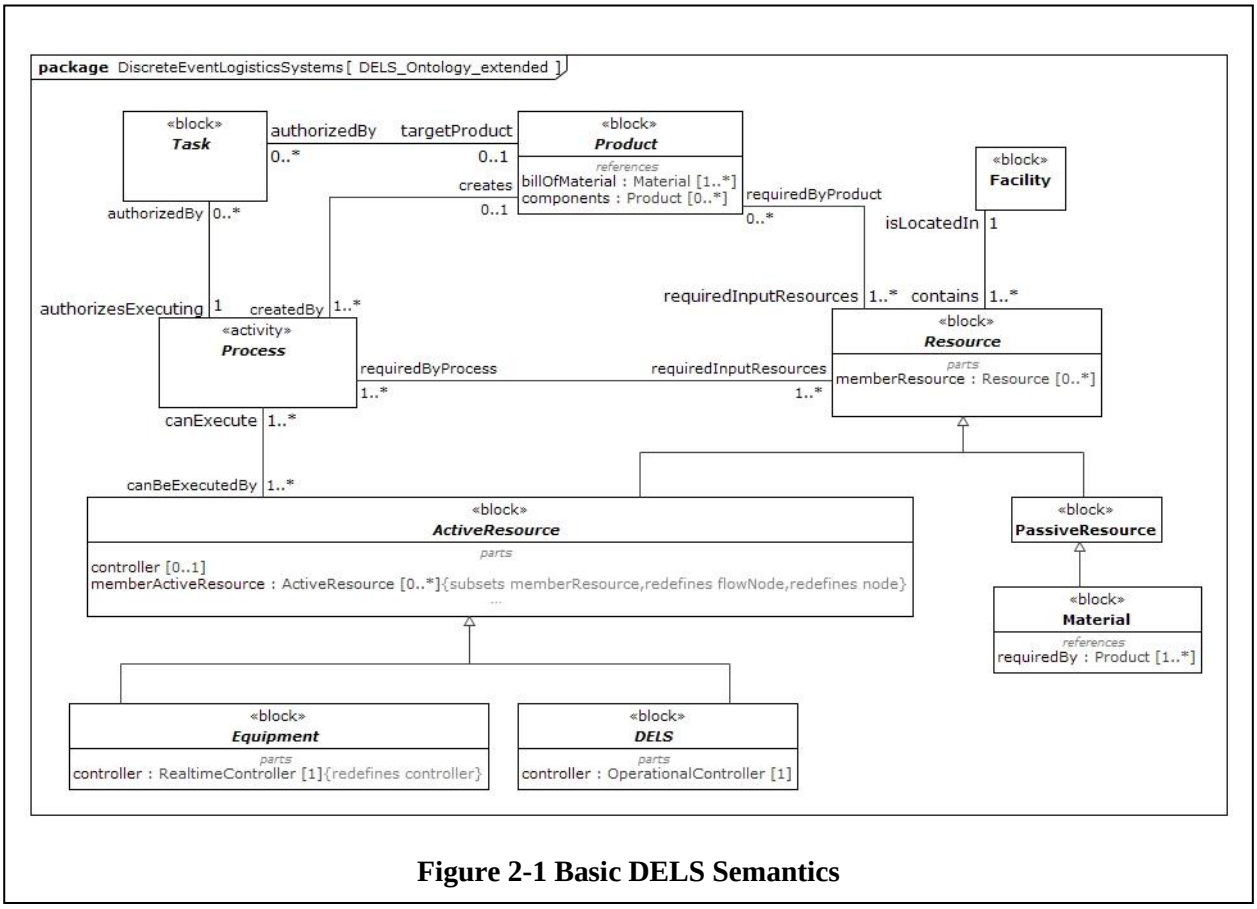

Figure 2-1 Basic DELS Semantics

In the DELS ontology, all the elements are abstract objects, i.e., they need to be specialized to specific object types, instances and executions in an application. A very important aspect of the DELS ontology is that every class within the ontology may have as parts other objects of the same class. Thus, a product nests products, a process has sub-activities that are processes, a resource may contain resources, a facility may contain facilities and a task may contain tasks.

In the DELS ontology *ActiveResource* has a controller part; a DELS has an OperationalController while an equipment has a RealtimeController. Figure 2-2 illustrates a conceptual model for the DELS OperationalController identified in Figure 2-1. There are six specific types of decisions made by the OperationalController:

- Admission: will a received or offered task be accepted?
- Sequencing: in what order will accepted tasks be executed?
- Assignment: if there are alternatives, to which resource will a task be assigned?
- Change State: when shall the state of a resource be changed, e.g., tooling configuration?
- Process Planning: an accepted Task may have both explicit and implicit sub-tasks; the controller must be able to identify these sub-tasks
- Routing: what process should be executed next?

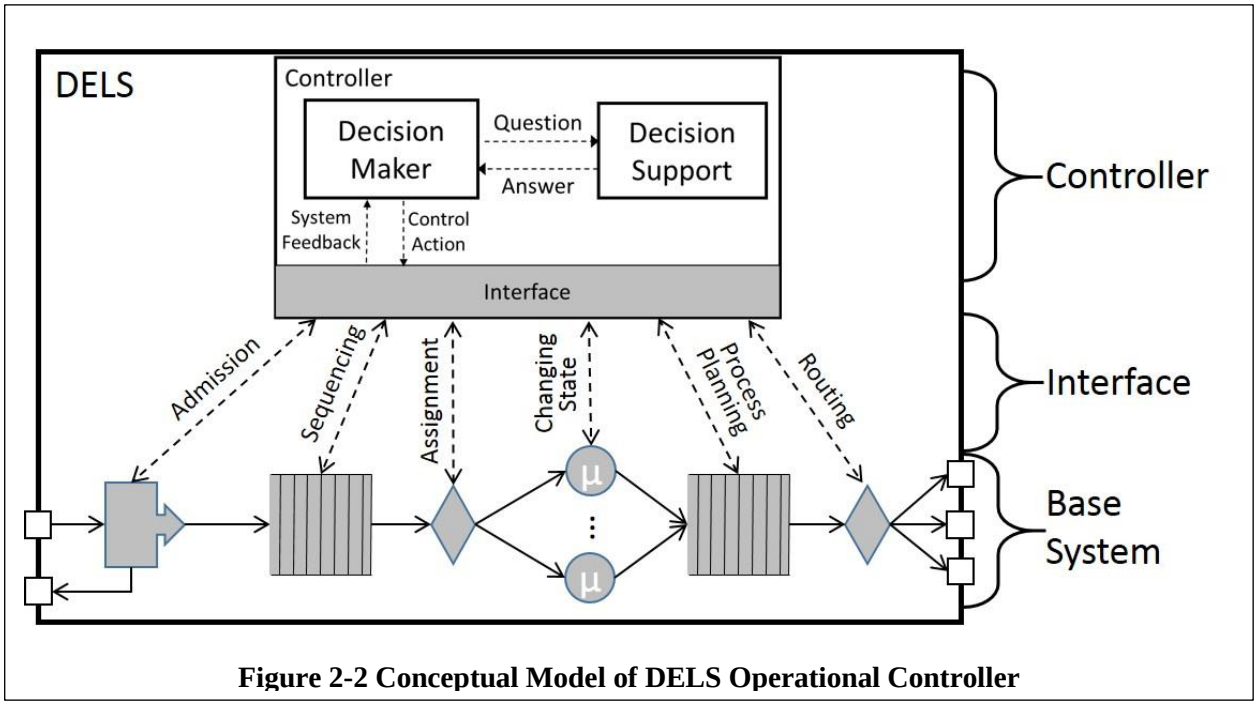

Figure 2-2 Conceptual Model of DELS Operational Controller

2.2 DELS to ISA-95 Correspondences

The DELS *Product* is a generalization of the ISA-95 *Manufacturing Bill* which has one or more associated *Product Segment*. Essentially, the *Product Segment* organizes the *Manufacturing Bill* according to the parts that are produced in the same or nearby locations within the same time frame. The DELS *Product* allows exactly the same kind of structuring for a specific product instance. This correspondence is illustrated in Figure 2-3below.

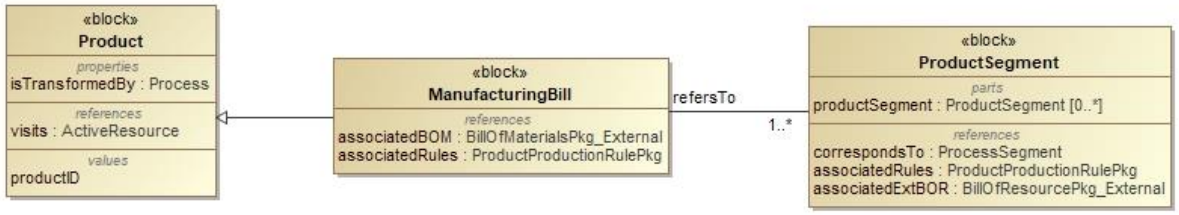

Figure 2-3 DELS Product and ISA-95 Manufacturing Bill

The DELS *Process* represents the information that typically is captured in process plans and detailed work instructions, thus *Process* generalizes *Product Production Rule*, *Production Routing* and *Process Segment*. *Product Production Rule* is a detailed instruction, thus can be modeled as a SysML activity. *Production Routing* is expressed through the precedence relationships in the DELS *Process. Process Segment* identifies the various resources required for a *Product Segment* and these are defined in *Process*, either as inputs or as the owner of the behavior represented by an activity. All of this information, in many instances, already exists, or is being developed in dedicated authoring systems, so it is conceptually straightforward to extract it to populate the *Process* model. These correspondences are illustrated in Figure 2-4below.

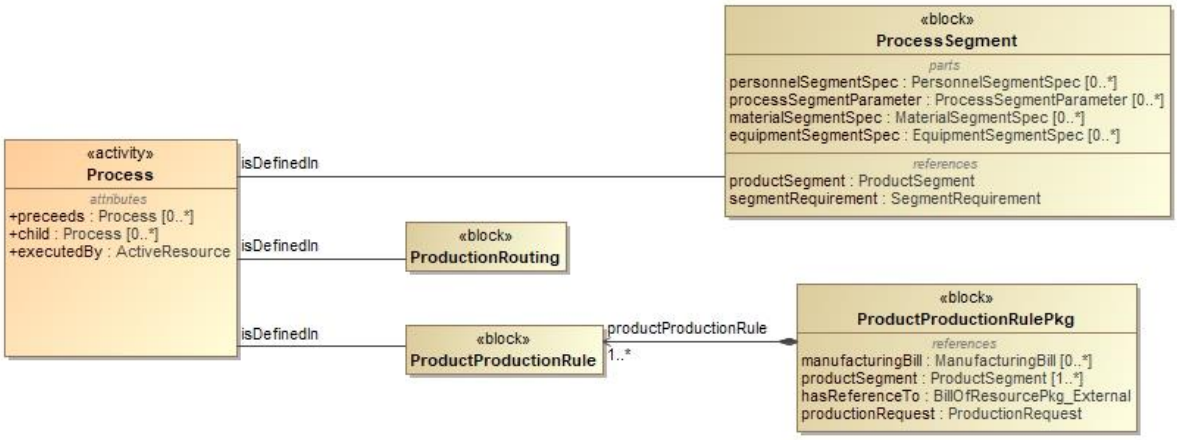

Figure 2-4 Process in DELS and ISA-95

Note also that *Product Segment* identifies a particular *Process Segment*. The corresponding DELS *Product* is created by the corresponding DELS *Process*.

DELS *Resource* models are in some respects very similar to the information contained in the ISA-95 *Process Segment* but are a generalization of the various resource categories identified in ISA-95. This is illustrated in Figure 2-5 below. Similarly, DELS *Facility* is a generalization of the specific ISA-95 categories of Enterprise, Site, Area, Process Cell, Production Unit, and Production Line, as shown in Figure 2-6.

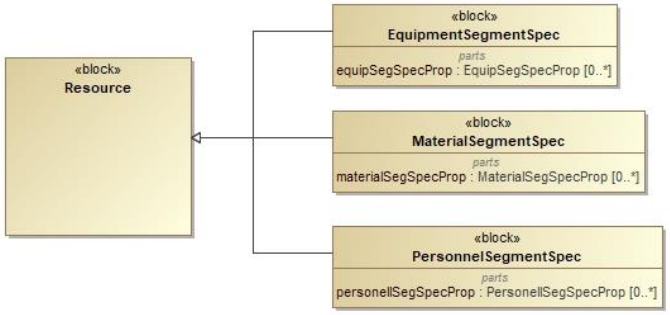

Figure 2-5 Resource in DELS and ISA-95

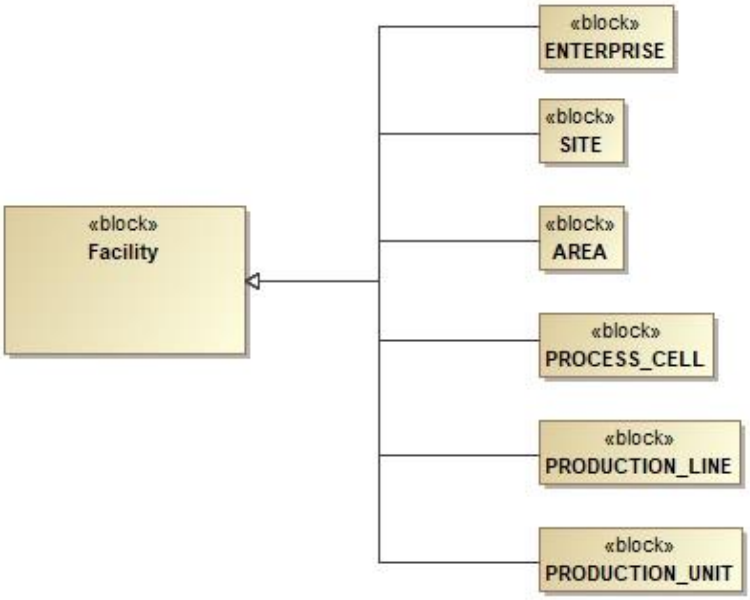

Figure 2-6 Facility in DELS and ISA-95

DELS *Task* is a generalization of the ISA-95 *Production Schedule*, *Production Request*, and *Segment Requirement* as shown in Figure 2-7. One significant distinction in the DELS framework is that material handling also is a *Process*, executed by a *Resource*, but one which may not be defined *a priori*, i.e., not defined in *Product Production Rule*. For example, a *Product* may need to visit a *Work Cell*, but at the moment it is ready to be moved, there is not space available at the target *Work Cell*, so the *Product* must be moved to a temporary storage location, and later retrieved and moved to the target *Work Cell*. A controller may need to create these kinds of *Task* even though they are not explicit in the process plan or work instructions.

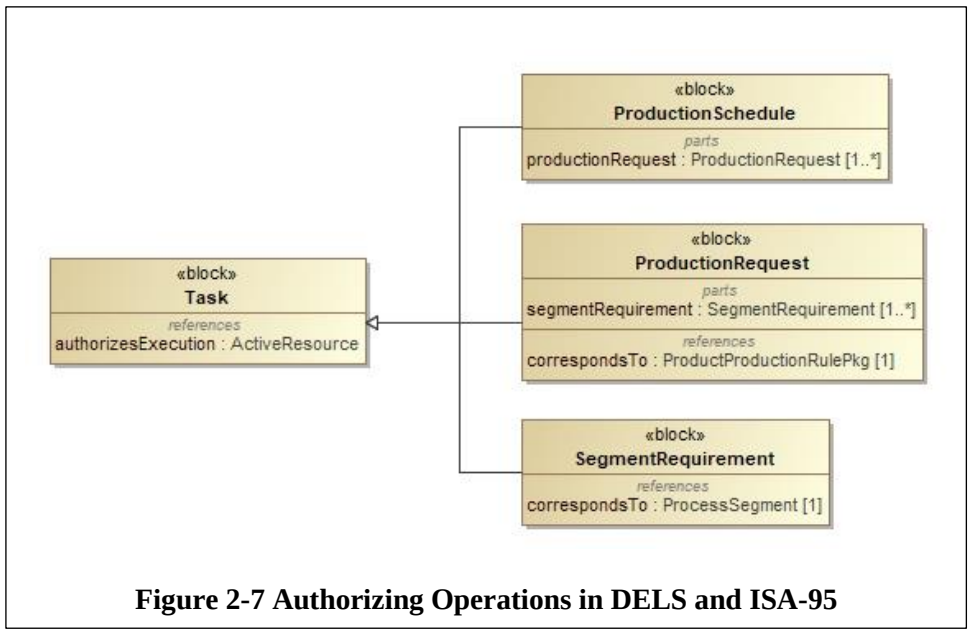

Figure 2-7 Authorizing Operations in DELS and ISA-95

In summary, there is a relatively straightforward mapping between DELS and ISA-95, except for controller objects. DELS specifically identifies controllers as a part of *ActiveResource*. Resources in the ISA-95 Level 3 domain have L3 controllers, and resources in the ISA-95 Level 2 domain have L2 controllers. The DELS ontology does not extend below the Level 2 controllers.

3. Operational Controller Function and Architecture

In the DELS semantic framework a discrete event logistics system consists of an operational (ISA-95 Level 3) controller and a set of active resources within the controller's domain, each having well defined capabilities. The controller is responsible for operational decisions regarding the admission of new tasks and the execution of existing tasks. This chapter explores the transition from the abstract semantics to a concrete implementation of a controller by addressing the control requirements, control functions, and their logical organization into an architecture.

The fundamental behavior of the operational controller is decision making, and as described in (Sprock, Bock, & McGinnis, 2019), there are five specific kinds of operational control decisions: admission, assignment, scheduling, routing, and active resource state change.

Figure 3-1 below presents a conceptual representation of the processes involving a DELS, its controller and its active resource set. Conceptually, new tasks are received by the controller, which may reject them. If they are accepted, upon completion the DELS controller reports the result. The active resources receive tasks from the controller. There may be a failure, or a DELS active resource may reject a task (if it also is a DELS). Otherwise, the process associated with the task is executed by some active resource, which reports the result to the controller. If the task completed by the active resource represents the completion of a task received by the DELS controller, the controller reports the result. Note that this is a conceptual representation; as one example, there may be multiple active resources, so a single port is not an accurate model. However, the model is a good starting point for discussing operational controller functions and architecture.

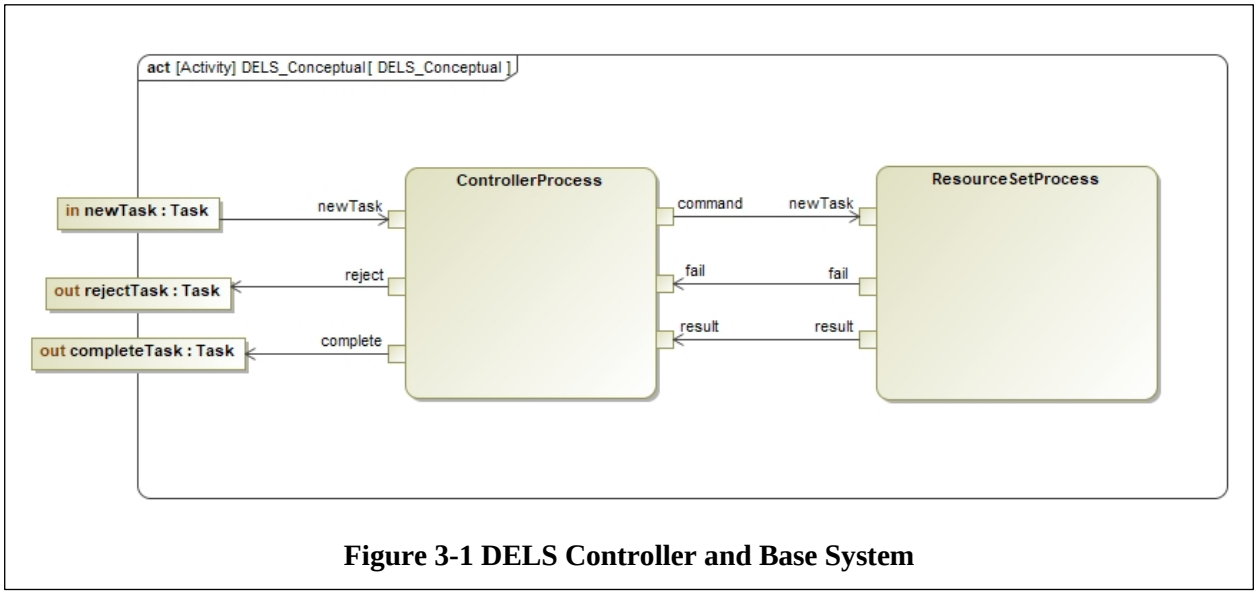

Figure 3-1 DELS Controller and Base System

Given this context, what can be inferred about the functions and architecture of the operational controller? Answering this question requires careful consideration of the definition of task, the mechanisms for triggering controller decision-making and the nature of the decision problems which the controller must resolve.

3.1 Requirement: Defining Task

The fundamental requirement for an operations controller is to manage the completion of tasks it has accepted recognizing performance objectives and task execution constraints. In (Sprock, Thiers, McGinnis, & Bock, 2019) task is described as a cyber-physical object representing both the authorization for process execution and the (passive) resource upon which the processes operate. In the discussion below, however, task will refer only to the authorization component, and "releasing" a task refers to transmitting the task to the resource that will execute the associated process. A task accepted by the controller may correspond to a process that can be directly executed, or it may correspond to a multi-step process, in which case the controller will release the individual sub-tasks. An accepted task is not completed until all its subtasks have been completed. The term "task" in the following discussion may refer to the task as accepted by the controller or to any of its constituent sub-tasks.

A task in the DELS context is the authorization of a resource to execute a process on a specific part or part set, or "job" for the purposes of this discussion. The job and the task may arrive together, for example a paper workorder in a bin of parts. Alternatively, the job may arrive before the task, inducing the resource to request a task, or the task may arrive before the job, requiring the resource to match a job to a previously received task.

In a manufacturing context, the task "produce product" corresponds to a process plan, which can be conceptualized as a directed acyclic graph (DAG) where nodes correspond to the specification of processes to be executed and edges correspond to precedence constraints (see, e.g., (Cho, Son, & Jones, 2006). The process plan describes the sequence of process steps that must be accomplished and as a consequence also identifies required capabilities of the set of resources that execute those process steps.

The DELS process that corresponds to a task can be represented using a SysML activity diagram, with call action nodes representing directly executable processes (process steps) and control flow paths representing precedence constraints. However, the task itself, the authorization for process execution, is not an activity, it is better conceptualized as structured data consumed by a controller and translated by the controller into invocations of active resource behavioral capabilities.

The traditional process plan for producing a product or service invariably specifies only the "make" processes. For example, the process plan for a circuit card assembly will specify the manufacturing process steps, such as clean the bare board, apply solder paste, place components, reflow, and inspect. However, in producing the circuit card assembly, there will be a number of "move" processes, and perhaps "store" processes as well. In addition, there may be contingencies that arise, such as failing an inspection, that require additional process steps. Without these additional processes, the circuit card assembly cannot be produced, but these processes are not spelled out in the process plan.

Because move, store and resolve processes are executed by active resources, their executions must be authorized by a controller. In some fashion, the "make"-focused task, e.g., make circuit card assembly, must be elaborated to include not only the individual make process steps but also subtasks for move, store and resolve processes. It is perhaps worth noting that a detailed "pedigree" for a product would include not only information corresponding to make process steps, but for every other process step. Thus, historical data could provide a template for an extended process model, and thus for an extended task model.

A fundamental issue in designing a controller architecture is choosing the specific mechanisms by which an accepted task is elaborated to generate the required move, store and resolve tasks. Knowledge of the implied elaborated process can be made explicit in some way and used as input to controller functions. Alternatively, knowledge of the extended process model could be incorporated implicitly in controller

function computational algorithms. The added sub-tasks might be created when the task is accepted, or they might be created on-the-fly as the subtasks are released. No matter how this issue is resolved, it should be noted that there likely always will be disruptions, i.e., process results which have not been anticipated, and which the controller cannot resolve based solely on the accepted task and predefined elaborations of the task.

3.2 Function: Operational Decisions

Recall that (Sprock, Thiers, McGinnis, & Bock, 2019) identify five operational control decision types: admission, sequencing, assignment, routing and state change. These decisions are implemented through the execution of resource behaviors, authorized by tasks. Further, (Sprock, Thiers, McGinnis, & Bock, 2019) categorize tasks as:

> "*availableTasks* (role played by Task) are tasks that have been accepted, admitted, and are waiting in the *availableTaskQueue* to be serviced. *completedTasks* are tasks that have been serviced and are stored 0 in an *completedTaskQueue* waiting to depart the system. *inProcessTasks* are tasks currently being served by the system and located in/at some *memberResource* (usually equipment)."

The tasks in the *availableTaskQueue* should be tasks that can be released to some active resource in the controller's domain, so they may represent sub-tasks of an admitted task. Only those tasks in the *availableTaskQueue* with no binding precedence constraints are candidates to be released immediately by the controller to authorize process execution. These will be referred to as *readyTasks*. Also, move tasks often will authorize a material handling DELS to execute a move process, i.e., the material handling DELS will determine, on-the-fly, exactly how to execute the authorized move, e.g., how to route the carrier.

The *availableTasks* have been admitted after considering both capability (does the DELS have active resources capable of executing the task?) and capacity (can the DELS complete the task in a timely manner without causing unacceptable delays to other already accepted tasks?). A direct consequence is that the controller must have access to information about the set of active resources in its domain, including their current capabilities, potential capabilities (realizable by a change state process) and current state with regard to ability to accept a task. This kind of information constitutes a *plantModel*, and a key functional requirement for the controller is maintaining this model.

The remaining operational decisions regarding *availableTasks* are assignment (which resource will execute the process associated with the task), scheduling (when will the process be executed) and routing (scheduling a supporting process not spelled out in the process plan). The controller also can decide to change the state (e.g., the setup) of an active resource. The specific timing and sequencing of these decisions may be different for different applications and implementations. For example, the assignment decision might be made when the task is accepted, or it might be delayed until the task is released. Routing decisions might be made when the task is accepted, but more realistically would be made at the time the controller considers a *readyTask*.

Clearly, in order to make operational decisions about tasks, the controller must have access to information identifying *availableTasks*, *readyTasks*, *inProcessTasks*, and *completedTasks*, i.e., to a complete *taskModel*. Thus another functional requirement for the controller is maintaining this model.

Operational decision-making requires queries to the *taskModel* to identify a set of *readyTasks*, and queries to the *plantModel* to identify the set of available resources for each *readyTask*. The assignment, scheduling and routing decisions depend upon the query results.

3.3 Function: Decision Triggers

A fundamental assumption about DELS is that they are event driven. Assuming the DELS is initialized with an empty *availableTaskQueue*, the events that might trigger operational decision-making are those that correspond to a state change for either the task list or some resource in the base system. These include: (1) arrival of a new task; (2) acceptance of a new task; (3) completion of some process execution authorized by a previously released task; and (4) resource failure. Other events, e.g., timers, might also trigger decision-making, but the focus here will be on the task and process related events. These change the states of the *plantModel* and *taskModel* and thus represent an opportunity to reconsider already made but not yet executed operational decisions or to make new operational decisions. There is little motivation for decision-making in between events since there has been no observable change in the state of the *plantModel* or *taskModel* since the most recent decision-making triggered by an event.

The DELS L3 controller must incorporate functions for detecting and interpreting events in its domain, as well as events associated with new task requests.

3.4 Controller Functions

From the previous discussion, the following controller functions can be identified:

- Maintaining *plantModel* and *taskModel*, i.e., updating as events occur and decisions are made (model maintenance)
- Querying *plantModel* and *taskModel* to support decision-making (model query)
- Identifying required processes not spelled out in traditional process plans (task elaboration)
- Detecting events and determining the appropriate response (event detection and interpretation)
- Formulating a decision-problem relevant to the kind of event, and perhaps to the query results (decision analysis problem formulation)
- Solving the decision problem (decision analysis problem solution)
- Translating the decision problem solution into assignable task(s) (task definition)
- Transmitting task(s) to assigned resources (task communication)

These eight functions are required in order for the operations controller to be able to make the five kinds of decisions identified in (Sprock, Bock, & McGinnis, 2019) and manage the corresponding tasks.

3.5 Controller Architecture

A conceptual controller architecture is illustrated in Figure 3-2. This is not claimed to be the only possible controller architecture, but it provides a reference point for discussing how controllers might be implemented in the DELS framework. All the functions identified above are included except task elaboration, because it is not clear where in the functional architecture it should appear. Resolving the issue of how to represent task elaboration requires a more in-depth investigation of process.

Parts visit active resources, where processes are executed on the part. Either explicitly or implicitly, the active resource must *induct* the part in order to execute a process and once the process completes, it must *discharge* the part. This can be conceptualized as a three-step process, a *get* operation, followed by the process, followed by a *put* operation. The sequence is *get-process-put*. The process can be either a make process, or a move or store process. In all cases, the fundamental sequence is the same.

The complete process for a product involves many of these *get-process-put* sequences as the parts and assemblies move from one active resource to another. In fact, it will alternate between make (or store) active

resources and material transport active resources. This is the case, regardless of the nature of the material transport resource—it can be an automated resource such as an AGV, or it could be a workstation operator who is moving the part to the next workstation.

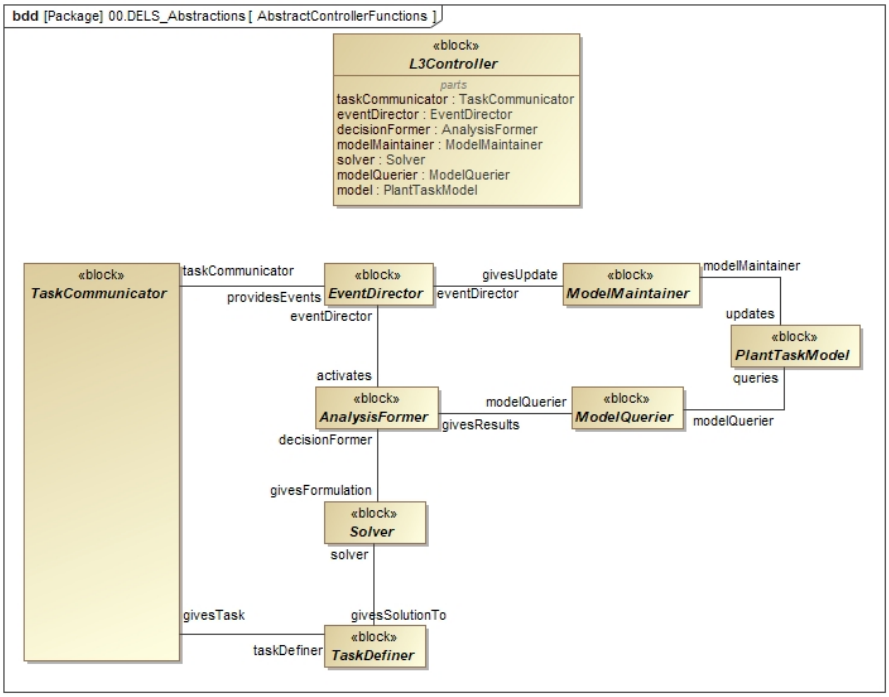

Figure 3-2 Conceptual Controller Architecture

The part must transition from one active resource to another. In the most general case, the arriving part to be processed is in some (input) interface location, from which it is retrieved, and once the process is complete, it is placed in some (output) interface location. The two locations could be the same. The interface location could be either a part or a reference for the active resource, i.e., it could belong to the active resource, or to some other active resource. Figure 3-3 illustrates one possible configuration. There are two workstations with "work positions" where their respective processes can be executed. Workstation1 has a robot that loads parts from the workcell induction location into the work position for workstation 1, moves the part from the work position for workstation 1 to the load/unload port for workstation 1, and finally moves the completed part from the load/unload port for workstation 2 to the workcell discharge location. The induction location is an interface location between the workcell and the active resource delivering parts to the workcell; similarly for the discharge location. The robot delivers and retrieves parts directly from the work position for workstation 1 but to a load/unload port for workstation 2. There is no separate interface location for workstation 1, but the load/unload port is the interface location for workstation 2. Workstation 1work position and workstation 2 load/unload port are reference properties for the robot.

The sequence of operations for a single part going through the workcell would be:

1. Get-move-put: robot moves part from induction location to workstation 1 workposition
2. (get)-make-(put): workstation 1 executes a process on the part, but the get and put are degenerate, as they will be performed by the workstation 1 robot.

3. Get-move-put: robot moves part from workstation 1 work position to workstation 2 load/unload port
4. Get-make-put: workstation 2 retrieves the part from the load/unload port, executes a process on the part, then puts the part to the load/unload port
5. Get-move-put: robot moves part from workstation 2 load/unload port to discharge location.

Note that the workcell also might include a store resource, to provide a temporary buffer between workstations 1 and 2, in case a part is ready to move from workstation 1, but workstation 2 is not finished with its current operation. Note also that a part cannot be placed in the workstation 2 load/unload port if workstation 2 is processing a part, as that would create a deadlock. This is, of course, a simple illustration of the get-move-put and interface location concepts.

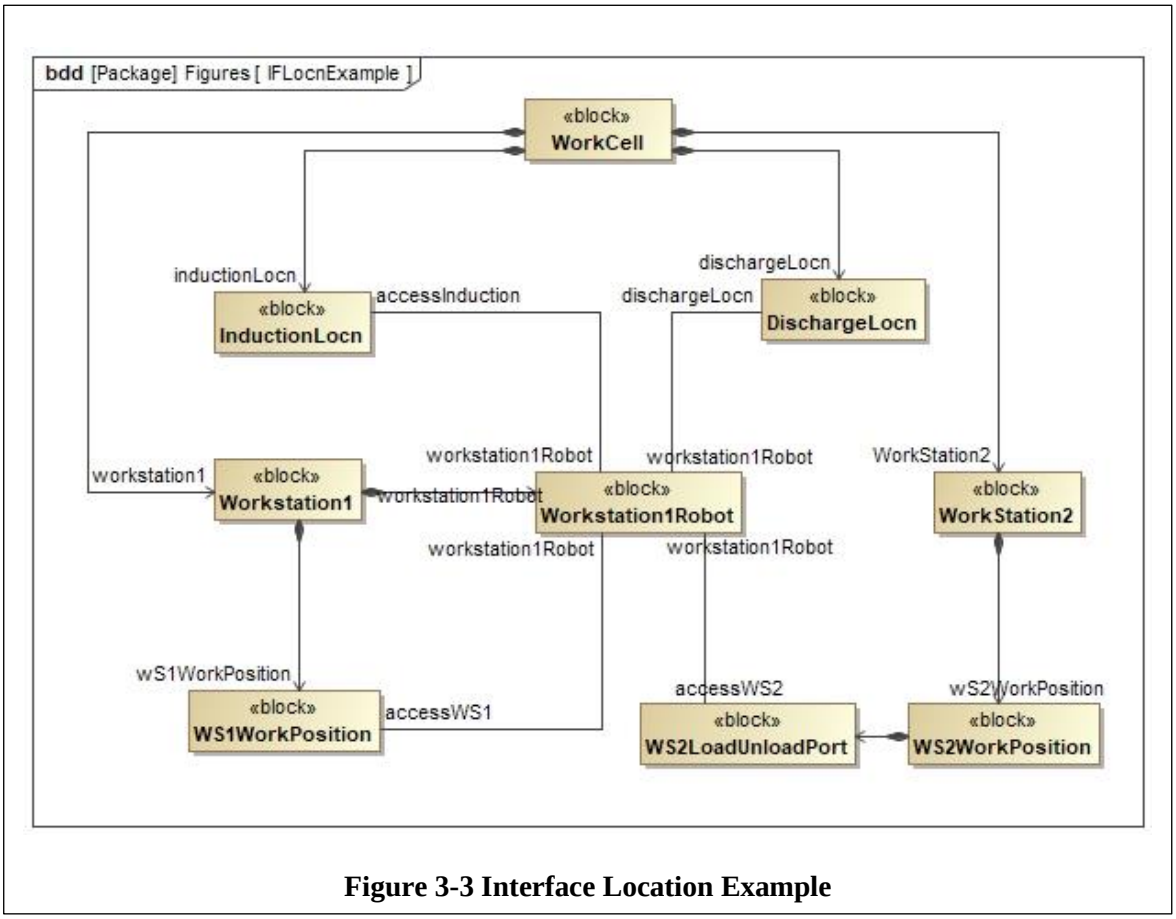

Figure 3-3 Interface Location Example

There are several important implications from this simple example. One is that the operations controller for the workcell must create tasks that essentially elaborate the process plan for parts going through the workcell to include all the move processes, but in addition, it must authorize, either explicitly or implicitly all the get and put operations. Depending upon the design of the system, move operations may require a destination interface location to be reserved in order to avoid deadlocks, a type of constraint that may not always be described as a simple precedence between tasks.

In the circuit card assembly example given earlier (see section 3.1) the task *assembleCircuitCard* is not directly executable. It must be elaborated into its constituent subtasks, which are placed on the *availableTaskList.* In fact, any task on the *availableTaskList* should be directly executable, i.e., it can be

assigned to some active resource in the controller's domain. Thus, the function of maintaining the task model is an appropriate place to integrate the elaboration of process plans to incorporate move processes, although the corresponding move tasks may not have a specific origin or destination, because the associated make tasks have not yet been assigned to active resources. When a make task has been assigned to a specific active resource, the origin for the subsequent move can be specified, but the destination may not be assigned until the make task has been completed.

For a given make, move or store task, the assignment to a particular active resource will dictate the associated get and put operations, i.e., whether or not they are degenerate. The corresponding get and put tasks can be placed on the *availableTask* list.

Since assigning a task to a resource is an event, it can trigger the event director, which can invoke the ModelMaintaner to update the *availableTaskList*. The conclusion is that the ModelMaintainer has two task elaboration functions: (1) an arriving task is elaborated into process authorizations which can be assigned to an active resource; and (2) whenever an *availableTask* is assigned to an active resource, details of previously defined move tasks are updated, and new get and put tasks are created as appropriate.

3.6 Summary

The key take-aways from this chapter are:

- Operations control invokes the behavior of active resources that make, move, store, or measure product
- The transfer of product between active resources occurs through an interface location and involves a *get-process-put* sequence of behaviors
- Control decisions are based on the state of accepted tasks and active resources in the controlled domain
- Control decisions are triggered by events.

4. Central-Fill Pharmacy Case Study

A central fill pharmacy (CFP) is "a pharmacy which is permitted by the state in which it is located to prepare controlled substances orders for dispensing pursuant to a valid prescription transmitted to it by a registered retail pharmacy and to return the labeled and filled prescriptions to the retail pharmacy for delivery to the ultimate user"[1]. The main advantages of a CFP include cost reduction, through inventory consolidation and improved resource utilization, and giving pharmacists in local pharmacies the flexibility to focus on customers. The disadvantage is the delay associated with sending a prescription to the CFP, and the transport cost and delay associated with the physical delivery to the local pharmacy or direct to the patient. This delay is not critical for "routine" refills.

A high-volume CFP (HVCFP) uses automation to speed the filling of prescriptions, further improving labor productivity and substantially reducing the cycle time and cost for fulfillment. Fundamental challenges in designing and operating a highly automated CFP include: (1) selecting the right portfolio of automation technologies; (2) designing the material handling automation to integrate the drug dispensing technologies; (3) assigning drugs to dispensing technologies, and perhaps configuring the technologies for operation; and (4) operational control to achieve goals regarding accuracy, cost, throughput and response time.

This case study identifies the "product" produced by a HVCFP, the processes required to produce that product, the resources used to execute the required processes, and the organization of resources into a facility. It also identifies the control functions of the HVCFP, using the ISA-95 standard architecture as a framework. The system description is based on a particular HVCFP architecture and the case study is for a specific instantiation of that architecture. However, the system model and simulation model contain elements that are reusable for other HVCFP architectures, although new model elements might have to be developed for those architectures.

4.1 Concept of Operation

A HVCFP will be capable of filling prescriptions, or "scripts," for many drugs, perhaps several thousand, and the demand rates for these drugs will differ significantly. Drugs are identified by their National Drug Code, or NDC. The NDC Pareto curve may be extremely skewed, with the top 2 or 3 percent of the NDCs accounting for 70% of the scripts filled. Generally, the most often dispensed drugs will be for controlling blood pressure or cholesterol level, or for diabetes.

The HVCFP receives orders via the internet from local pharmacies. In one operational protocol, these orders may be transmitted at any time; orders received when the HVCFP is not operating are accumulated for the following day and available orders not completed in one day are carried over to the next day. All orders from a particular customer (pharmacy) completed during the HVCFP daily operation will be accumulated for delivery overnight. Other protocols are possible, such as guaranteeing that orders received before a designated cutoff time will be filled on the day received and delivered overnight. In another scenario orders may arrive with due dates, and can be filled and delivered earlier. There are many possible variations.

A given HVCFP will serve a large number of local pharmacies, perhaps several hundred. The populations served by these local pharmacies may differ demographically, and if so, their ordering patterns may be quite

[1] 21 CFR 1300.01 (44) [Title 21 Food and Drugs; Chapter II Drug Enforcement Administration, Department of Justice; Part 1300 Definitions]

different. The HVCFP will see significant day-to-day variability in the volume of scripts and mix of drugs ordered. The average volume and the mix of drugs ordered also may change depending on the season, and even the time of the month. The total number of scripts to be fulfilled, as well as the mix of drugs will change over time, driven by both demographic changes and advances in medicine.

4.2 HVCFP Product

A HVCFP produces batches of patient-specific orders ready for delivery to the originating local pharmacy. At regular intervals, e.g., the end of each working day, these batches will be loaded into a delivery vehicle for transport to the local pharmacies. In a particular pharmacy's batch, each individual patient's order will consist of one or more NDCs in appropriate packaging—vials for pills, bottles for liquids, and various unit-of-use packages. These NDCs must be packaged together, along with necessary paperwork, for delivery to the individual at the originating pharmacy.

Each NDC can be characterized as "dispensable" or "manual". Scripts for dispensable drugs—typically in pill form—can be filled using automation, whereas scripts for manual drugs must be filled by a human operator. The latter might correspond to liquids to be measured, items that come prepackaged in a form not suitable for automation ("unit of use"), drugs requiring refrigeration, etc. An order for a particular patient may include both dispensable and manual drugs.

4.3 HVCFP Processes

There are three phases of operation in a HVCFP. The first phase is the dispensing of individual drugs. The second phase is assembling individual patients' orders, which may consist of multiple scripts. The third phase is assembling all the orders for a specific pharmacy and delivering them. Note that dispensing drugs to fill a script and accumulating scripts to complete an order are referred to as the "fulfillment" function of the HVCFP. Accumulating the orders by pharmacy is referred to as the "delivery function". There are four fundamental processes for dispensing a drug, four fundamental processes for completing an individual's order, and three fundamental processes for completing a pharmacy's delivery.

To dispense a drug (dispense phase):

1. Prepare an appropriate container into which the drug will be dispensed. If the drug is prepackaged as a "unit of use" this process is not required.
2. Dispense the drug. This involves counting pills, measuring liquids, or retrieving a unit of use.
3. Verify the drug and quantity. In HVCFP facilities, this process is a regulatory requirement to insure patient safety.
4. Seal the container. Once a drug has been dispensed and verified, the container must be sealed. For prepacked unit-of-use drugs, this process is not required.

To complete an individual's order (order accumulation phase):

1. Accumulate all scripts for the individual order. Individual scrips in the order may be filled using different technologies, but all scripts must be brought together to complete the order.
2. Add each script's container to customer-specific packaging, typically a plastic bag.
3. Add drug-specific instructions, required notifications and other documentation.
4. Seal individual order.

To complete a pharmacy's delivery (pharmacy accumulation phase):

1. Accumulate all the individual orders for the specific pharmacy.
2. Seal the pharmacy order
3. Deliver the pharmacy order to a shipping dock, load into a delivery vehicle and transport to the pharmacy.

Of course, there are other related processes in a HVCFP. For example, inventory processes for storing drugs prior to use in filling orders, refrigeration processes, replenishing automation, etc. However, this case focuses specifically on the operational processes in supplying individual orders to pharmacies.

4.4 HVCFP Resources

Over the past twenty years, a number of drug dispensing automation technologies have been developed, and HVCFP solutions are offered by several suppliers (see, e.g., http://www.computertalk.com/feature-stories/cover-story-september-october-2014-the-evolution-of-central-fill or http://www.mckesson.com/about-mckesson/our-company/businesses/mckesson-high-volume-solutions/) .

4.4.1 Dispense Phase Resources

For dispense phase processes, there are essentially four kinds of automation resources. The first is automation resources for dispensing, labeling, weighing and capping vials, which [will] contain pills. For example, dispensing, labelling and weighing empty vials may be combined into a single automated process, where the weighing determines a tare weight, used later to verify the dispensed pill count. There also may be stand-alone resources for weighing or capping vials.

Second, there are resources that can automatically dispense a specific number of pills from a drug-specific canister into a vial. The automation technologies for dispensing drugs in pill form essentially use gravity to remove pills from an inverted canister, along with a mechanism that counts the number of pills dispensed and stops the flow when the required number of pills have been dispensed. There are two variations of this technology, which might be termed "high speed" and "high flexibility". A high speed resource will receive empty, but labeled and tared vials transported in a "puck" on a conveyor, see Figure 4-1. The puck will be moved under a dispenser, the vial filled with pills, and then moved in the puck to stations for verification, weighing, and capping. The vial never leaves the puck. A single dispensing machine might have, say, six dispensers, and machines can be "ganged" together to provide a multiple of six dispensers, all served by the same puck conveyor. Clearly, high speed dispensing technologies require considerable integration of all the individual resources and the puck conveyor, but can be very effective for dispensing drugs for which there is a high demand rate.

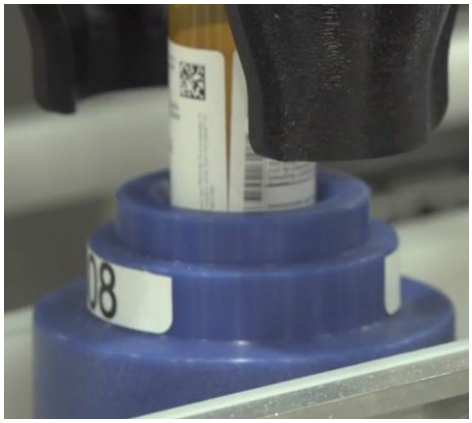

Figure 4-1 Vial in Puck

http://www.mckesson.com/pharmacies/mail-order/central-fill/

A high flexibility resource operates quite differently. It is essentially a robotic workstation, which may have as many as 200 or more canisters, or pill types. Labeled and tare weighed vials may be delivered to the workstation via pucks and the vials removed from the pucks by the robot. Alternatively, the workstation may have its own capability to dispense, label and tare weigh vials. Figure 4-2 shows a robot holding a vial under a dispensing canister. For high flexibility workstations with vial dispensing capability, the filled vials are dropped into totes moved on a tote conveyor. There can be multiple high-flexibility workstations, as well as manual fill stations integrated via the tote conveyor. This technology can be effective for drugs that are ordered often enough to keep the robot reasonably busy. Extremely rarely ordered drugs are probably most economically handled manually.

As a comparison, a high speed technology might be capable of fulfilling 18 different drugs, each at a rate of 3 scripts/minute (for a total of 54 scripts/minute) and a capital cost of $175,000, while a high flexibility technology might be capable of fulfilling any one of 200 different drugs at a total rate 2 scripts/minute and a capital cost of $200,000. These are illustrations only, and do not define the complete range of automated solution capabilities and costs.

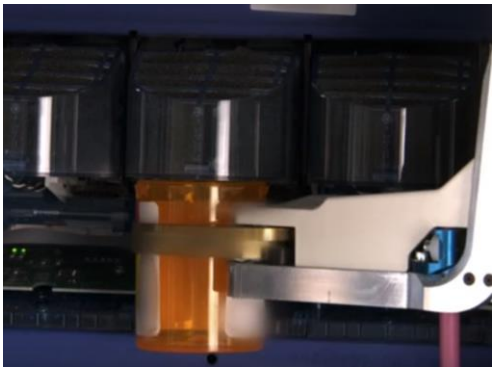

Figure 4-2 Robotic Workstation

https://www.youtube.com/watch?v=cBUigyLA0xg

The third kind of technology for the dispense phase is visual verification, which involve imaging the dispensed pills in the vial, and having the image verified by a pharmacist.

Finally, there is transport technology for moving vials between dispensing processes. For high speed dispensing or high flexibility dispensing with separate vial dispense, label and tare weigh, the pucks must be transported between the various stations or technologies that are performing the necessary dispensing, labeling, and other processes. For high flexibility dispensing with built-in capability for vial dispense, label and tare weigh, the robot provides all the needed transport of the vials between operations.

4.4.2 Order Accumulation Phase Resources

In the order accumulation phase, there are four basic technologies. There is the technology of accumulating the scripts in a customer order, which is accomplished by delivering the scrips to a bagging station that can, itself, be completely manual, partly automated or completely automated. The scripts can be delivered to the bagging station via a puck conveyor, if all the scripts in the order are filled from the high-speed technology. When some scripts are filled from manual workstations, or from the high-flexibility technology workstations with built-in vial dispense, label and tare weigh, then they are delivered to bagging in a tote via a tote conveyor, and they are accumulated in the tote as the tote travels to each dispense workstation along the tote conveyor. There is a special case of orders, called “combo orders” that have at least one script filled from the high-speed technology, and one script filled manually or from tote-based high-flexibility technology. In this case, one or more scripts filled from the high-speed technology, and contained in a puck must be transferred to the tote containing the rest of the scripts for the combo order. This can be accomplished by a vial transfer station (VTS), a robotic cell that removes vials from pucks, places them in temporary storage, then when the target tote is available, retrieves the pucks and deposits them into a tote, where the other items in the order are already accumulated.

4.4.3 Pharmacy Accumulation Phase Resources

Multiple orders from a given local pharmacy may be filled throughout the day and appear somewhat at random in the stream of orders coming from the fulfillment system. These orders must be accumulated into a pharmacy-specific container for delivery. At any time during the day, the stream of orders being delivered to the order consolidation process might contain orders for any of the customer pharmacies. Thus, the order consolidation process requires some form of sortation and accumulation by pharmacy. In a HVCFP processing tens of thousands of orders per day, automation is likely to be required, and there are a number of technological alternatives. All of them, however, have a similar functional form, i.e., the bagged orders are oriented on a conveyor, scanned to determine the destination pharmacy, conveyed past accumulation "lanes" and individual bags are diverted to the lane corresponding to their destination pharmacy. Figure 4-3 illustrates a portion of such a system, where there are accumulation lanes on either side of the sortation conveyor. An accumulation lane is assigned to a particular pharmacy and orders for that pharmacy are discharged from the sort conveyor and fall into the bin. When the bin is full, it is sealed and moved to a staging are prior to loading into a delivery vehicle.

When the number of pharmacy customers is large, in the hundreds, it may not be possible to dedicate an accumulation lane to each pharmacy; rather some operational policy may be necessary to permit accumulation lanes to be shared by multiple pharmacies. This policy may address both the way that orders are released for fulfillment and the way the sortation/accumulation system is managed. A fundamental challenge with shared accumulation lanes is that bagged orders for a pharmacy may arrive to a lane that is shared, but not currently assigned to that pharmacy. This possibility dictates the need for a "sort error" lane, where such sort failures can be accumulated for later disposition. This disposition may involve running these orders through the sorter again, if there are many, or perhaps manually sorting them, if there are not so many.

Figure 4-3 Order Sortation System

https://www.youtube.com/watch?v=VdHcq0_zq_M

4.5 Facility

The order fulfillment resources in the HVCFP are organized logically into seven subsystems:

- High speed dispense system that employs a range of fast dispense resources, as well as resources for dispensing and tare weighing vials, verifying dispense quantity and NDC, capping and bagging
- Puck conveyor system that provides all product movement through the high speed dispense system
- High flexibility dispense system that consists of a range of automated and manual workstations to dispense drugs that either cannot be automatically dispensed, or are ordered often enough to justify automated dispensing, but not often enough for high speed dispensing
- Tote conveyor system that provides all product movement through the high flexibility dispense system
- Vial transfer system that moves vials from the high speed system to the high flexibility system

- Take-away conveyor system that removes bagged orders from both high speed and high flexibility systems
- Order sortation/accumulation system that sorts orders by ordering pharmacy

4.6 Operational Control

The control processes can be distinguished according to the ISA 95 standard, which is illustrated in Figure 4-4. In level 3, manufacturing operations management, decisions will be made such as the release of orders to the fulfillment process, or management of the sorting capabilities. Level 2 is where operations management decisions are translated into execution by production resources, e.g., controlling the movement of a package on a conveyor, based on level 3 routing decisions. Deciding when a manual station should be staffed and by whom would be an operational decision. Lower level control decisions focus on data acquisition, and managing the execution of predefined behaviors in automation. In this case study, the focus is on operational control, i.e., level 3.

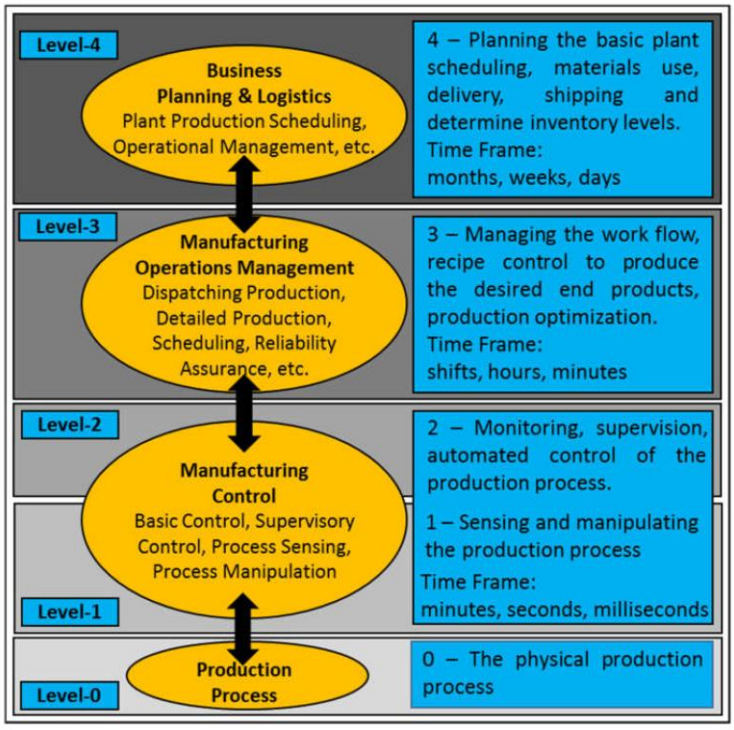

Figure 4-4 ISA-95 Control Hierarchy

http://www.mdpi.com/1999-5903/9/3/35/htm

For the HVCFP, the level 3 control decisions will include:

- Whether to accept an offered order
- When to release each of the scripts in an accepted order for fulfillment

- If multiple resources are available to execute a given fulfillment process, the assignment of process step to resource
- If a resource must be re-configured to execute a process, when to change the resource configuration

It is important to be able to judge whether or not a control system is performing well. In the case of the CFP, criteria might include:

- Maximum achievable rate of order fulfillment, measured in scripts per hour
- Distribution of fulfillment cycle time, from order release to bagging
- Resource utilization distributions

Other criteria might be defined. In general, what a system owner will care about is system cost (investment and operating), system service level (fraction of accepted orders filled on time), and capacity margin (room for demand growth).

4.7 System Summary

A HVCFP represents a significant investment and promises significant operational cost savings over stand-alone local pharmacy operations. Realizing these potential savings depends on making good decisions about the selection of technologies, planning, and executing operations for a system that has many individual components, and large amounts of data related to capabilities, capacities and daily demand. This is an ideal setting in which to apply the principles and methods of Model-Based Systems Engineering (MBSE).

5. Model-Based Systems Engineering for the HVCFP[2]

A fundamental goal in applying MBSE is to provide a single "source of truth" for the system definition to be used by the various decision-makers involved in designing, planning, managing and controlling the system of interest, in this case a central fill pharmacy. In order to do this, there must be a shared definition of the system that includes all the information needed by those decision makers. Thus, a common semantic model or *reference model* is required for the system model. A common practice in developing and applying such reference models is a layered approach, as in (Sprock, Thiers, McGinnis, & Bock, 2019). At the highest level of abstraction is the fundamental "language" being used, in this case it is OMG SysML™. Using, e.g., SysML, an upper level ontology applicable to a broad range of systems is defined, in this case, Discrete Event Logistics Systems, or DELS. The DELS ontology can be refined for a subset of DELS that is central fill pharmacies, or CFPs and then further refined for the subset of CFPs that is high volume CFPs, or HVCFPs. Finally, the lower level ontology can be used to create a specification of the particular system of interest, a particular HVCFP. This chapter addresses the upper level abstractions that provide the reference model for use in developing a system model for a particular CFP. The following chapter will use these abstractions to create a detailed system model for the kind of CFP described in chapter 4.

5.1 Upper Level Ontology—Generic CFP

Since the CFP is an example of a discrete event logistics system (DELS), we can employ the semantics defined for DELS (Sprock, Thiers, McGinnis, & Bock, 2019). Figure 5-1 expands on the DELS reference model presented in chapter 2. The terms "product", "process" and "resource" have the meaning as defined earlier for the HVCFP, but the DELS reference model and Figure 5-1 add semantics. Resources can contain other resources (memberResource). As an example, the High Speed Fill System is a resource, and it contains a number of other resources such as vial dispense and labeler, etc. As SysML blocks, resources can be connected to one another, and this is how a model represents potential for product to flow through the system.

"Process" defines how a product is produced. Because process is modeled as a SysML activity, it can contain other processes (activities). As an example, the process for filling a prescription can be modeled using process models (activities) for dispensing and labeling a vial, dispensing pills, and capping the vial, and precedences between these processes can be modeled using control flows. In general, the "Process" will describe a generic way of producing a product. For example, if there are multiple resources capable of executing a particular production step, the "Process" model typically will not specify which resource is to be used; rather, that decision is made by an operational controller.

Figure 5-1 adds two objects to the DELS reference model, Controller and PlantModel. A controller issues the tasks that authorize execution of processes, and it uses data from a plant model in order to make decisions about which tasks to issue and when to issue them. Clearly, the controller must have, either implicitly or explicitly, knowledge about the processes required to produce a product, the resources within its control domain and their process capabilities. It should be noted that while a task authorizes the execution of a process, the task is received by the active resource that has the capability for and actually executes the process.

[2] All SysML diagrams come from the HVCFP.mdzip model

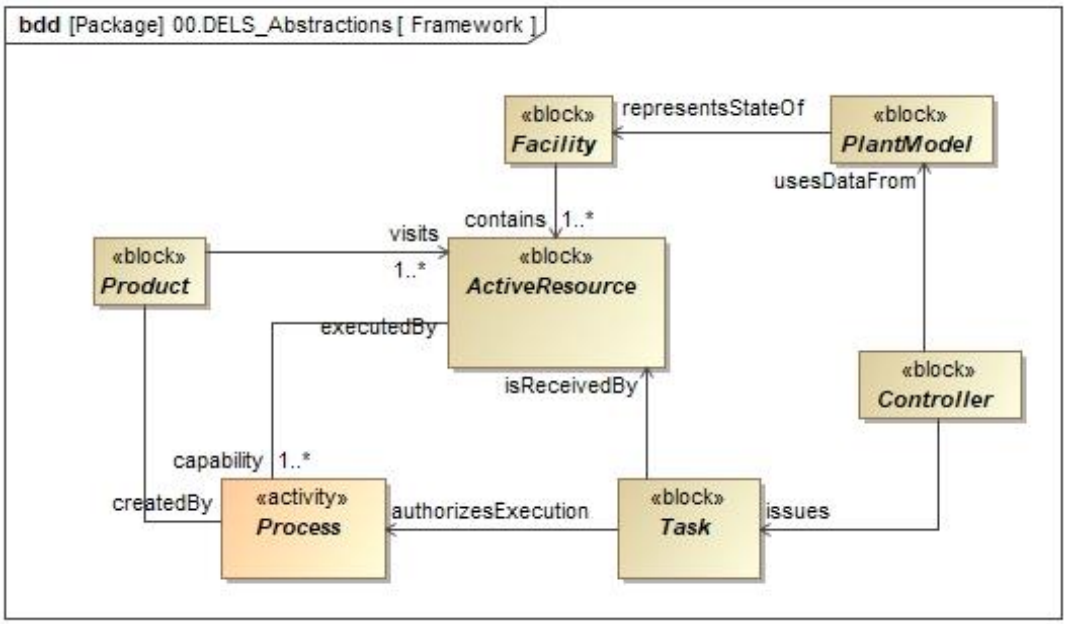

Figure 5-1 Common Semantics for Discrete Event Logistics Systems

A DELS controller makes decisions that manage the operations of the DELS resources. According to (Sprock, Bock, & McGinnis, 2019), there are essentially five kinds of decisions the controller may need to make:

- Admission: whether or not to accept a task
- Sequence: the sequence of task execution
- Assignment: the resource to be used to execute a task
- Routing: identifying the next process to execute
- Setup: changing the capability state of a resource

Not every DELS will make all five kinds of decisions, as will be illustrated in the HVCFP.

The remaining sections of this section will present key elements of a SysML-based system model for the class of central fill pharmacies described in chapter 4.

5.2 CFP Context

The CFP serves a population of local pharmacies, or "stores" by fulfilling individual customer orders, consisting of one or more prescriptions. The CFP, in turn, is served by a population of suppliers, who provide the drugs that are dispensed by the CFP. This is illustrated in Figure 5-2. This study focuses on the customer order fulfillment only; the replenishment processes and resources are not considered.

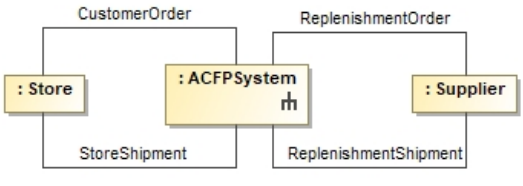

Figure 5-2 CFP Context

5.3 Defining Product

The "product" for the CFP is the batch of fulfilled customer orders to be delivered to a particular pharmacy at the end of a particular day. This product, identified in Figure 5-3 as one or more *StoreTote*s, aggregates the *BaggedOrder*s for individual *CustomerOrder*s, which themselves contain the *NDCPackage* for each script defined by an *OrderLine* in a *CustomerOrder*. Thus, producing the CFP "product" requires first fulfilling the customer order lines, then assembling them into bagged orders, which fulfill the individual customer order, then assembling bagged orders into store specific totes to fulfill the store order.

There is not a "bill of materials" for a generic store order. In fact, not all the customer orders in a particular store order typically will be known at the same time. Some customer orders from the store may come in overnight, while others may come in during the day. If the CFP specifies a "cutoff time", i.e., a time after which received customer orders are not promised for delivery overnight, then that cutoff time is the first point at which the CFP might know all the customer orders in a particular store order. It is conceivable that in the lifetime of the CFP, it will never produce two identical store orders for any store.

Similarly, there is not a "bill of materials" for a generic customer order. Rather, a customer order consists of some number of drugs, chosen from the catalog of drugs that can be provided by the CFP. In a given day, the CFP might produce several identical orders, e.g., 90 doses of Benecar 20 mg. However, there also will be many unique customer orders, consisting of multiple drugs. The generic bill of materials simply states that a customer order consists of one or more lines, and each line specifies an NDC and a quantity or amount.

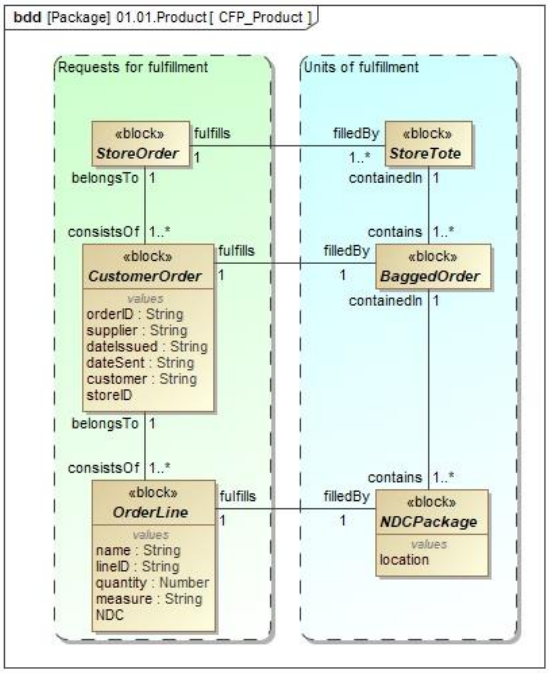

Figure 5-3 CFP "Product"

5.4 Defining Process

Process models specify the transformations required to create the product, in this case, the store orders. For the definition of a *Process* to be useful, the DELS must contain some *Resource* with a behavior that can be invoked by a *Task* to execute the *Process*. A *Process* may be associated with an aggregate *Resource*, i.e., a *Resource* that nests other *MemberResource*s, in which case it will be an aggregate *Process* and will need to be refined into its constituent *Process*es executed by the corresponding *MemberResource*s.

It is worth noting that the number of distinct "products" (*StoreOrder*s) of the CFP is practically infinite. Considering just individual *CustomerOrder*s, the number of unique orders is basically:

$$\sum_{j=1}^{j=N} \binom{M}{j}$$

where M is the number of different drugs available in the CFP and N is the maximum number of scripts in a customer order. Note that M is typically in the range of several thousand and a practical limit for N is perhaps five; the number of different 5-script orders is greater than 2.65×10^{14}. Thus, it should be clear that a Process is not going to be defined *a priori* for every possible customer order.

The generic process for filling a customer order is shown in Figure 5-4. Order lines can be filled in any sequence. After they are filled, they are accumulated. Once all the lines in an order are accumulated, the order is bagged. In general, the number of lines is not known *a priori*.

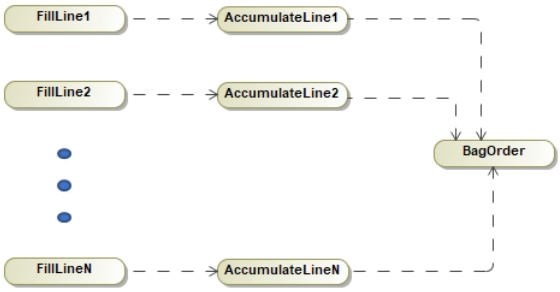

Figure 5-4 Generic Order Fulfillment Process

Since a process is not created explicitly for each individual customer order, it must be created "on-the-fly" by the CFP operational control system. Note also that while the process for filling a line may be precisely defined, there can be "exceptions" requiring special processing, e.g., incorrect pill count or failed capping. This also must be handled by operational control on-the-fly.

Although there is no single "order assembly process" there is structure for the process of producing an order, as illustrated in Figure 5-5. Producing a customer order requires filling each of the associated order lines, accumulating each of the order lines, bagging the order and sorting the order to a store tote. The definition of the process for filling order lines will differ with the nature of the drug—is automated dispense possible—and the nature of its packaging—is it a unit of use item, or must it be dispensed into a generic container? However, as with order filling, there is a generic structure, as shown in Figure 5-6. Every order

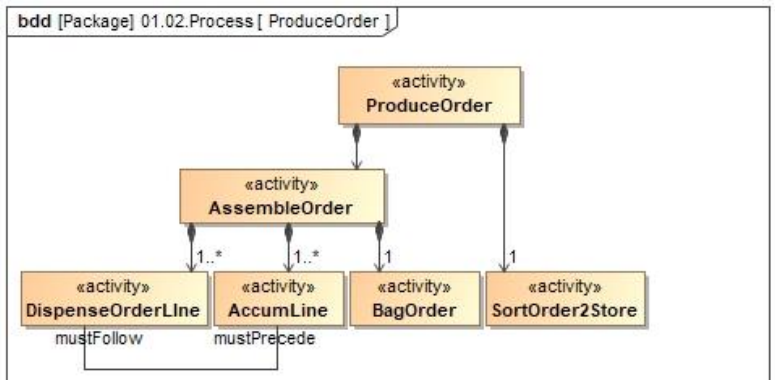

Figure 5-5 Order Filling Structure

line must be dispensed in some way, verified for drug and quantity, and accumulated. But dispensing a package (e.g., a vial), labeling, tare weighing and sealing are only necessary for drugs that are measured or counted from a bulk quantity or amount into an order specific quantity or amount.

In the context of Figure 5-2, the CFP itself has the capability to produce a StoreOrder. It does that by first assembling customer orders (in bags) and then assembling store orders (by sorting the customer orders). This process is summarized in the activity diagram of Figure 5-7. Both the order assembly and store order assembly processes must be further refined, recognizing the structure illustrated in Figure 5-5 and Figure 5-6. Figure 5-7 shows a stream of bagged orders going directly from the order assembly process to the store order assembly (sorting) process. The implication of this is that there is an actual handoff of the "bagged order" between these two systems, and therefore that the associated material handling is part of one of the two systems. If that material handling is a separate system, then the material handling process also should be included in the activity diagram to support object flow between the two assembly processes.

With figures 4.5-4.7, we have the abstract definition of the fundamental processes in assembling customer and store orders as well as the abstract definition of CFP processes. Further fining and elaborating these processes requires identifying the CFP resources that will be used and how they will be controlled. This discussion will be deferred until resources have been defined.

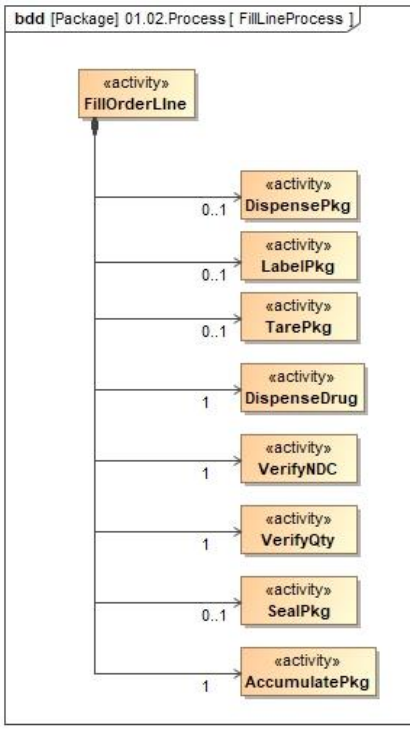

Figure 5-6 Line Filling Structure

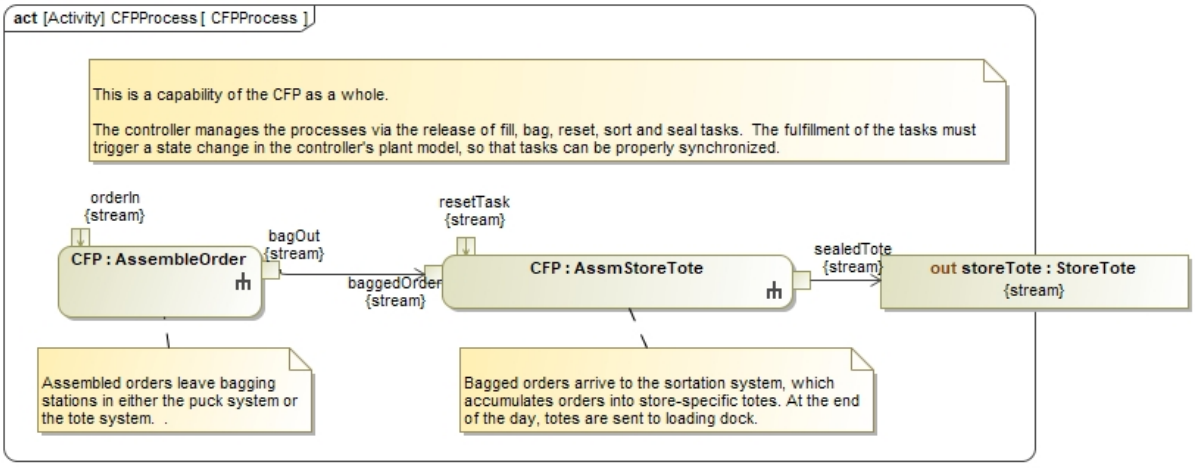

Figure 5-7 CFP Fulfillment Process

5.5 Defining Resource

In the DELS framework, active resources have capabilities to execute processes, and thereby create products. Resource models must identify active resources, their capabilities, their *memberResources* if any, and their process capabilities. Central fill pharmacies will utilize some generic and some specialized resources. These resources must be carefully defined to support system design and operational decision making. In particular, resources must be characterized in terms of *capabilities*, i.e., the processes that they can execute, and *capacities*, i.e., the throughput rates for those processes when executed. Additional attributes may be needed as well, for example, footprint and required clearances, utility requirements, maintenance schedules, etc.

Figure 5-8 illustrates the main subsystems of a CFP and their key components. There is an *OrderFillsystem* with multiple fulfillment workstations, and a *SortSystem* with multiple sort lanes. The details of these resources will differ with different technologies for dispensing drugs and sorting bagged orders, but the fundamental system architecture is as shown in Figure 5-8. These are *active* resources.

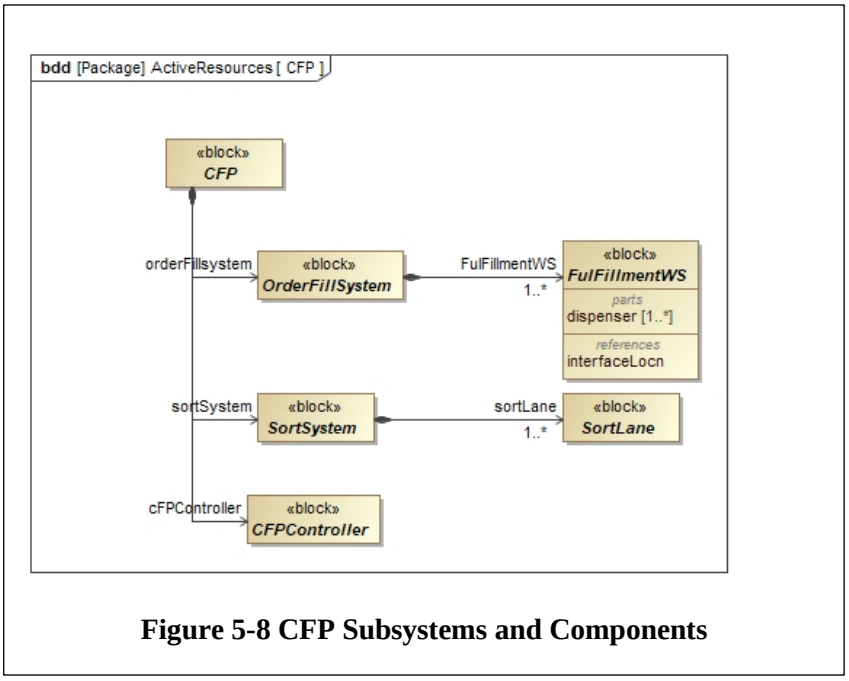

Figure 5-8 CFP Subsystems and Components

A CFP also uses a number of passive resources, primarily packaging for dispensed drugs or carriers for transported order lines or orders being assembled. These include but may not be limited to bottles, boxes, vials, pucks, and totes.

5.6 Defining Control

At the level of the CFP, there are only two control decisions, Admission and Sequencing. A CFP controller will have three part properties representing aspects of a plant model: an *AdmittedOrderSet* is a set of admitted customer orders, a *Batchsize*, or the number of orders released to the *OrderFillSystem* at one time, and a *TaskSet*, representing authorizations for the order fill system to execute processes necessary to fill customer orders. The CFP controller makes an admission decision about customer orders, it batches

customer orders for fulfillment, and *it translates customer orders into fulfillment tasks*, i.e., into authorizations for processes that actually produce filled customer orders.

Figure 5-9 illustrates the controller process for admitting a new customer order. The input to the process is the *CustomerOrder* being evaluated. There is an *AdmissionDecision* which returns a Boolean; if the value is 1 the new order is added to the *AdmittedOrderSet* and if 0, the new order is rejected. The details of implementing the decision algorithm and the information processing associated with the *AdmittedOrderSet* will depend on the details of the target application.

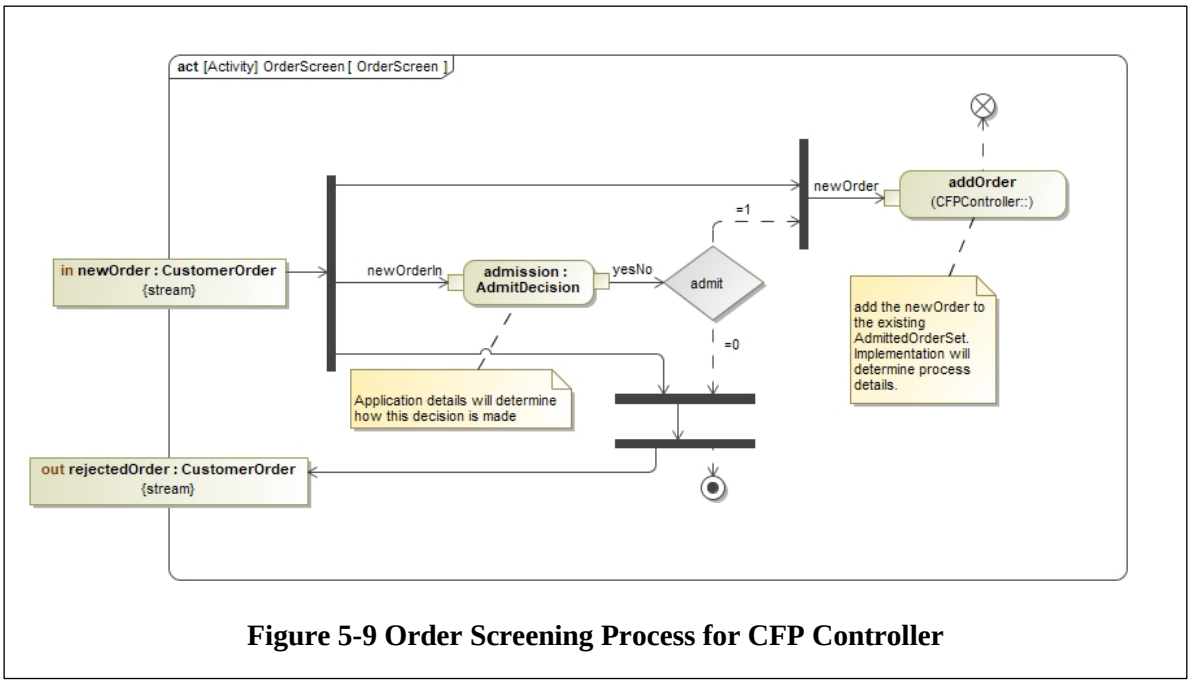

Figure 5-9 Order Screening Process for CFP Controller

The CFP controller also will release batches of orders for fulfillment, which means it must select a subset of all the accepted orders to release in a batch. The decision process for selecting the orders to batch together may well depend upon the state of the fulfillment resources, in terms of already released workload, as well as the overall operational strategy for the CFP. Thus, it is reasonable to expect that the batching decision is made with recognition of the tasks corresponding to the selected orders. In other words, between the admission decision and the batching decisions, customer orders are translated into tasks. The structure of the tasks is illustrated in Figure 5-10 but it is important to note that the details of translating orders to tasks will necessarily depend upon the specific process steps required in a particular implementation. In particular, *DispenseTask* may involve several steps for pill dispensing. Also, there will be material handling tasks associated with moving pucks or totes among the various fulfillment workstations.

In a similar fashion Figure 5-11 illustrates the sequencing process for the CFP controller, which conforms to the definition in (Sprock, Bock, & McGinnis, 2019). The method for computing the sort index will depend on the application. The orders selected will be the first *batchSize* orders in the sorted *TaskSet*.

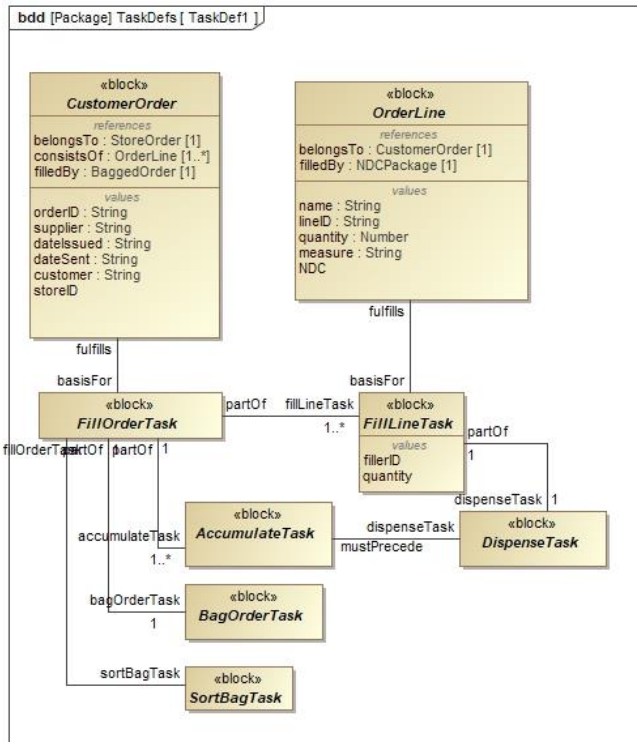

Figure 5-10 Task Structure for CFP

For a generic CFP, the Assign decision is degenerate—all orders go to the *OrderFillSystem*—and the Routing decisions are made within the *OrderFillSystem*. Also, there is no operational StateChange decision,

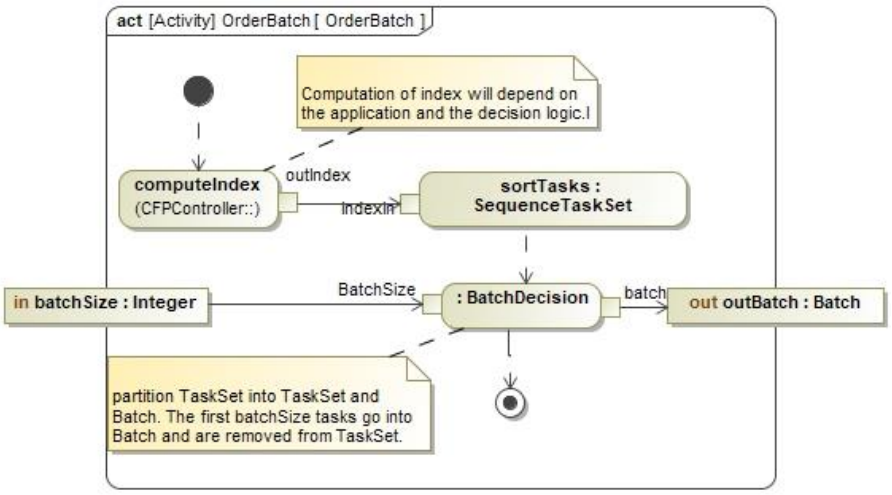

Figure 5-11 Order Batching Process for CFP Controller

the most relevant state change would be the NDC assigned to a particular dispense workstation, and that is more a planning decision than an operational decision.

5.7 Summary

This chapter provides the basic semantics for a central fill pharmacy. These semantics must be elaborated, e.g., for specific resource types, specific process steps and specific control processes and decisions. This elaboration will be illustrated in the following chapter, using the example CFP from Chapter 4 as the specific application.

The key take-aways from this chapter are:

- Adding task and controller to the basic DELS framework
- Defining the CFP context
- Defining CFP product in terms of requests for fulfillment and units of fulfillment
- Identifying the fundamental challenge of explicit fulfillment process models and suggesting a response to this challenge by defining the structure of fulfillment processes
- Identifying the CFP-level processes and resource organization
- Identifying CFP level order screening and batching
- Identifying the task structure for the CFP

6 Demonstration CFP System Model

In this chapter, the semantic definitions in Chapter 5 are extended and applied to the Demonstration CFP (subsequently abbreviated to *DemoCFP*) described in Chapter 4 to create a detailed system model specifying products, processes, resources, facilities and operational control. The intent for this system model is to provide a system specification sufficient to support the development of a discrete event simulation model for testing alternative decision processes for the various DELS and equipment controllers.

The context and product definitions from Chapter 5 serve adequately to describe the context and products for *DemoCFP*, recognizing that there are several hundred stores being served, around 2000 NDCs being dispensed, and approximately 30,000 orders processed each day.

An important consideration in developing detailed models for large-scale complex systems is the organization of the model itself. Poorly organized SysML models are difficult to navigate effectively, thus difficult to validate and difficult to understand. At this writing, there is not a legacy of production system SysML models from which to derive best modeling practices. The practice that will be followed in presenting the *DemoCFP* model is illustrated in Figure 6-1. The primary organizing concept is *active resource* and in general an active resource is defined in a package of the same name. The SysML package *DemoCFP* captures the specification of the *DemoCFP*. The *DemoCFP* package contains four other packages, one each for modeling the product or service produced by *DemoCFP*, the resources contained in *DemoCFP*, the process capabilities of *DemoCFP*, and the control of *DemoCFP*. In this case the *DemoCFP* has four part properties which are themselves DELS and thus are modeled with the same package structure. The *DemoCFPController* also has its own package, which contains three additional packages where decisions, control processes and task definitions are modeled. The decision package addresses the five types of decisions for DELS controllers. The process package addresses all the non-decision processes required by a controller, such as maintaining information about tasks or invoking behaviors of owned resources. The task definition package defines the tasks which the controller can assigned to owned resources.

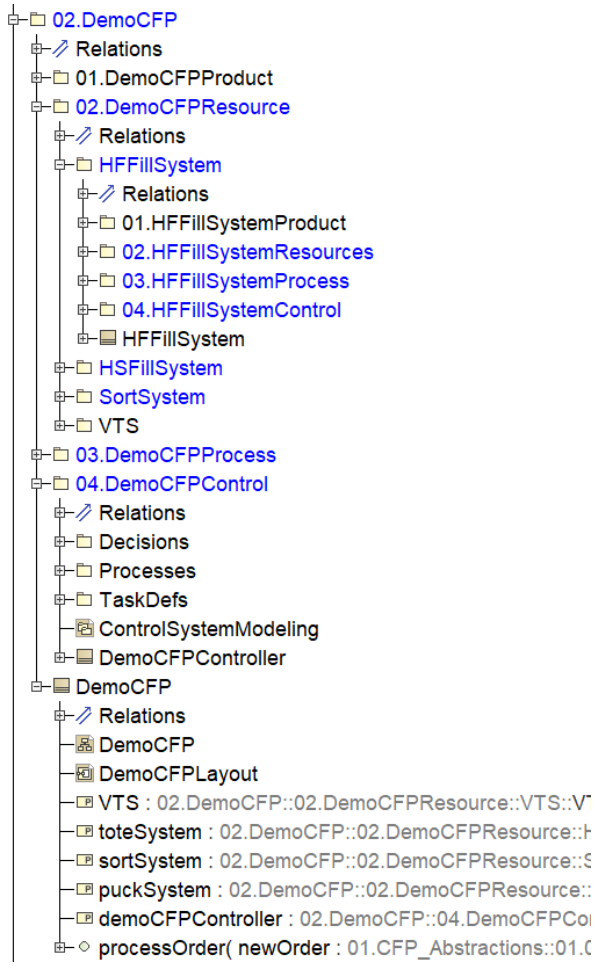

Figure 6-1 System Model Organization

The sequencing of packages within an active resource package is in the easiest order of presentation and explanation. Products and resources need to be identified to support process models, and some control models are best understood after resources and processes have been defined. It is the case, however, that in any interesting system model, there will inevitably be "forward references" in describing the products, resources, processes and controls.

The general modeling principles are:

- Any active resource or controller is modeled in its own package. Exceptions can be made for active resources having no owned resources and whose process model and operations can be modeled within the active resource block (for an example, see the robot arm of the *VTS*)
- An active resource package contains other packages specifying products/services produced, process capabilities, owned parts, and controls
- A controls package contains other packages specifying decisions, non-decision control processes, and task definitions for called behaviors.

6.1 *DemoCFP* Package

The *DemoCFP*, as shown in Figure 6-1, has five part properties and one operation. The five part properties represent the four subsystems—the high speed fill system, the vial transfer system, the high flexibility fill system and the sort system—and the CFP controller. In this model, *DemoCFP* has no value properties, but there could be value properties for site-specific information, such as location, etc.

6.1.1 *DemoCFP* Product

As shown in Figure 6-2, the *DemoCFP* is capable of producing two products, a *CFPCustomerOrder* which is a kind of *CustomerOrder*, and a *CFStoreOrder* which is a sealed tote containing customer orders from a particular store.

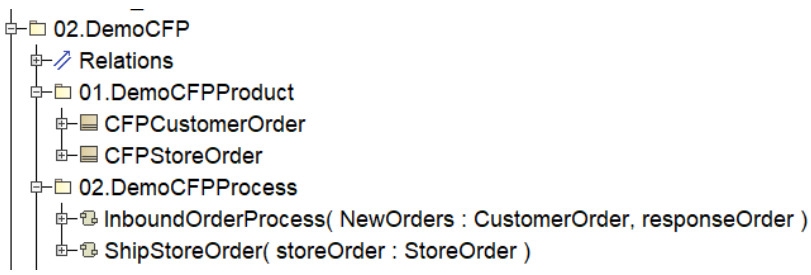

Figure 6-2 *DemoCFP* Product and Process

6.1.2 *DemoCFP* Resource

DemoCFP is composed from four active resource subsystems as illustrated in the block definition diagram in Figure 6-3, which also shows the major owned resources of the subsystems. The four major subsystems are:

- *HSFillSystem*, the puck conveyor based system with dedicated dispensers
- *HFFillSystem*, the tote conveyor based system with shared dispenser automation and manual dispensing

- *VTS*: the vial transfer systemn that bridges between the tote and puck systems, and
- *SortSystem*: the system that takes bagged orders from *HSFillSystem* and *HFFillSystem* and assembles the store totes.

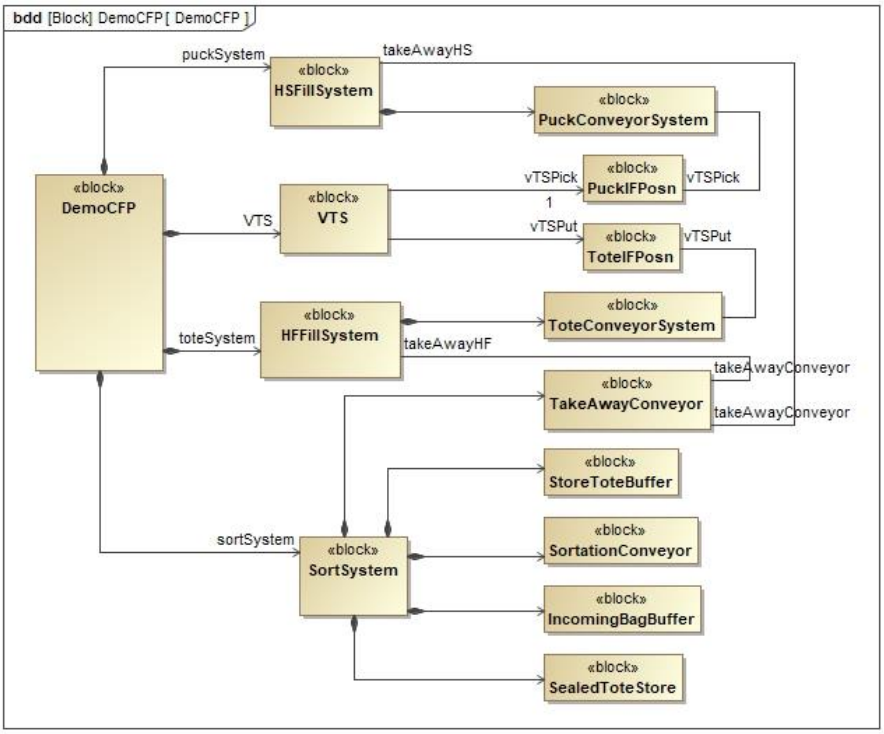

Figure 6-3 *DemoCFP* System Structure

The *vTSPick* and *vTSPut* reference properties for *VTS* are associations of the *PuckConveyorSystem* and *ToteConveyorSystem*, respectively, and are the locations where the *VTSRobot* can access pucks and totes. In a less precise representation, the *HSFillSystem* and the *HFFillSystem* both have the *TakeAwayConveyor* as a reference property, because their respective bagging stations deposit bagged customer orders on a segment of the *TakeAwayConveyor*. The *PuckConveyorSystem* is a part property of the *HSFillSystem* and *ToteConveyorSystem* is a part property of *HFFillSystem*. There are many interfaces between the conveyor systems and the parts of the fill systems, corresponding to locations where pucks or totes can be delivered to dispensing workstations. These will be defined in more detail when the subsystem models are presented.

The fifth major component of *DemoCFP*, not shown in Figure 6-3 is the controller, *DemoCFPController*.

The *DemoCFP*'s four major subsystems—*HSFillSystem*, *HFFillSystem*, *VTS*, and *SortationConveyor*— are DELS because all four systems, at some point, require logistical decisions to be made, and thus have L3 controllers.

The internal block diagram of Figure 6-4 corresponds to the block definition diagram in Figure 6-3 and shows the flows among the major subsystems. The details of each of the four major subsystems will be presented in subsequent sections.

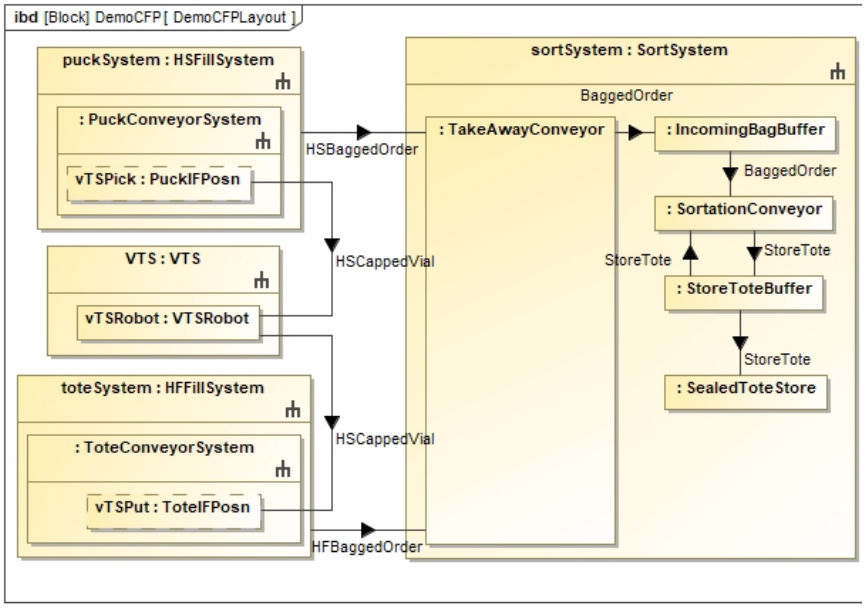

Figure 6-4 Flows in *DemoCFP*

Vials which are part of a combo order flow from the *PuckIFPosn* to the *VTSRobot*, and from the *VTSRobot* to the *ToteIFPosn*. Bagged orders flow from both *HSFillSystem* and *HFFillSystem* to the *TakeAwayConveyor*; from there to the *IncomingBagBuffer*, after which the sorting process takes place. Partially full *StoreTotes* are removed from the *SortationConveyor* when their lane is reassigned and placed into a temporary *StoreToteBuffer*. Full *StoreTotes* are sealed and put into a *SealedToteBuffer* to await loading into a delivery vehicle.

6.1.3 *DemoCFP* Process

DemoCFP has two process capabilities that are relevant in its context, one corresponding to the processing of incoming customer orders and one corresponding to delivering store orders at the end of the day.

The *InboundOrderProcess* represents the admission/rejection of a customer order. It is exposed to stores via the *processOrder* operation listed in Figure 6-1. An admitted order will subsequently be fulfilled by other processes invoked either by the *InboundOrderProcess* or by processes of the *CFPController* and become part of a *CFPStoreOrder*. The *ShipStoreOrder* process is not explicitly exposed to the stores but represents the *DemoCFP*'s contractual agreement to ship store orders the end of the day.

The *InboundOrderProcess* is shown in Figure 6-5. Orders are received via the internet at any time, and if the *DemoCFP* is not operating, these orders go into a buffer. When the *DemoCFP* is operating, customer orders are considered in first-come-first-served sequence and evaluated to see if all the order lines can be fulfilled, i.e., if the requested NDCs all are available. If not, the order is rejected, otherwise it is accepted. In either case, there is a response to the submitting store. An admitted order is evaluated and assigned an order type—*HSFillSystem* only, *HFFillSystem* only, or combo order—and given a wave assignment

corresponding to the store's assigned wave. The augmented order is added to a database of open orders, the *OpenOrderTable*, which will be used by subsequent *DemoCFP* control processes.

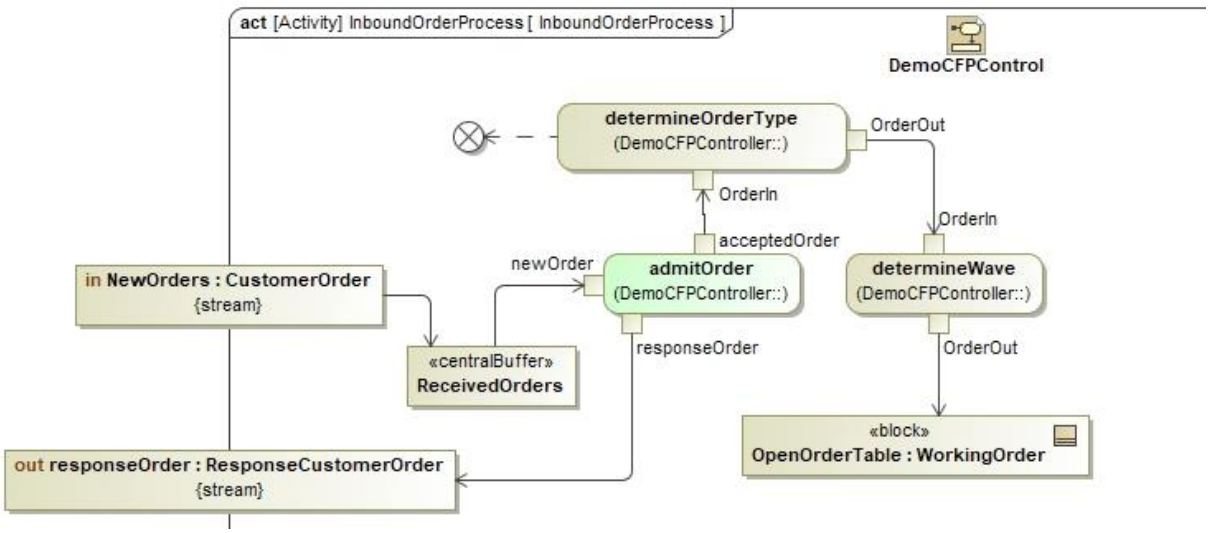

Figure 6-5 Inbound Order Process

The *ShipStoreOrder* process has not been modeled.

6.1.4 *DemoCFP* Control

DemoCFP control, up to the bagging of orders is summarized in Figure 6-6. The *DemoCFPController* has two key operations management functions. Orders are screened for admission and prepared for fulfillment by the *InboundOrderProcess* and released to the two fulfillment systems by the *BatchRelease* . Further control within the fulfillment resources is required to fill customer orders. Once orders are bagged, there is subsequent control to sort them to store totes which is not modeled here. Note that *fillPuckBatch* is a called behavior of the *HSFillSystem* and *fillToteBatch* is a called behavior of the *HFFillSystem*.

In the *InboundOrderProcess*, only the order admission constitutes a *decision*, and even that is a relatively trivial decision. The other actions are essentially administrative processes of the *CFPController*.

BatchRelease selects a batch of orders from the *OpenOrderTable* to release for fulfillment and converts the orders in this batch into tasks suitable for release to the two fill systems. This process is illustrated in Figure 6-7. The actions highlighted in green represent control decisions: *selectWaveSet, selectReleaseBach, sequence* represent schedule or sequence decisions, and *assignDispenser* represents a resource assignment. The *fillToteBatch* call operation action uses an exposed process of the *HFFillSystem* and the *fillPuckBatch* call operation action uses an exposed process of the *HSFillSystem*. The remaining call operation actions in the figure are data preparation functions performed by the *DemoCFPController*. The activity uses two accept event actions corresponding to the two fill systems completing all the currently released orders and requesting a new batch.

.

Recall from Figure 6-4 that order lines from a combo order that are filled in the *HSFillSystem* must go through the *VTS* to be combined with lines filled in the *HFFillSystem*. This requires some synchronization between the two systems. Part of the synchronization is realized in Figure 6-7, where for a given release batch, the combo lines are sequenced first for the *HSFillSystem* and last for the *HFFillSystem*. If it happens that a required vial has not yet reached the *VTS* by the time the corresponding order tote arrives, then an error has occurred, and the tote simply recirculates to try again, until the vial has arrived.

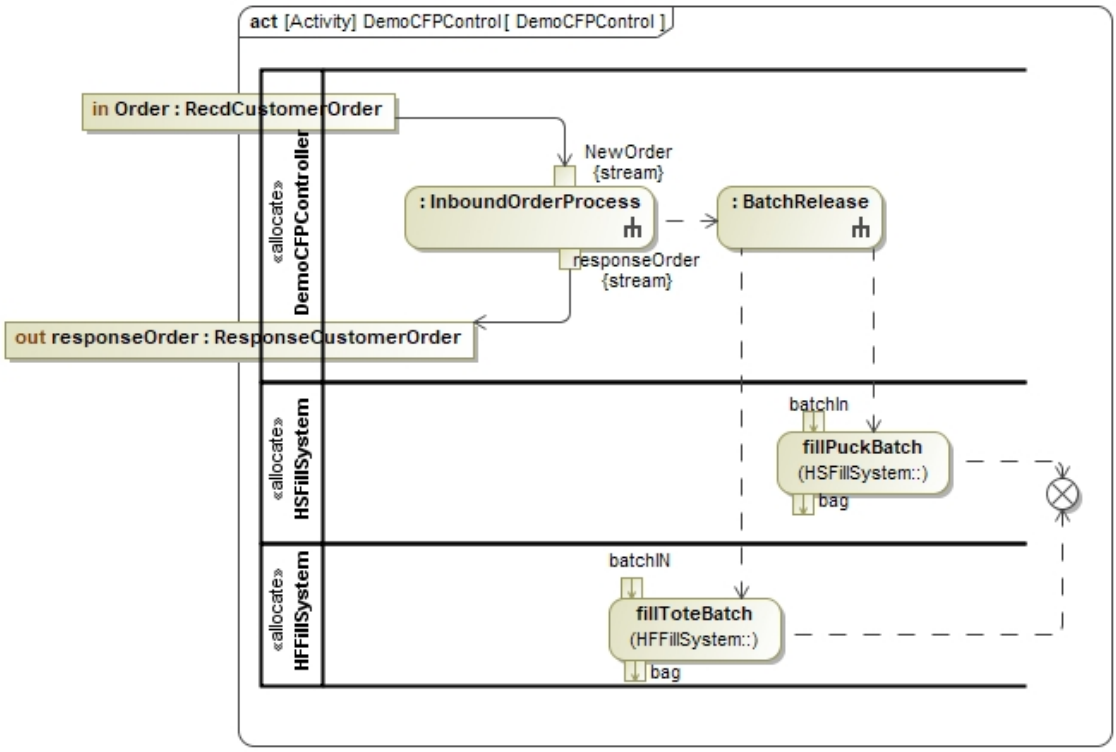

Figure 6-6 *DemoCFP* Control up to Bagging

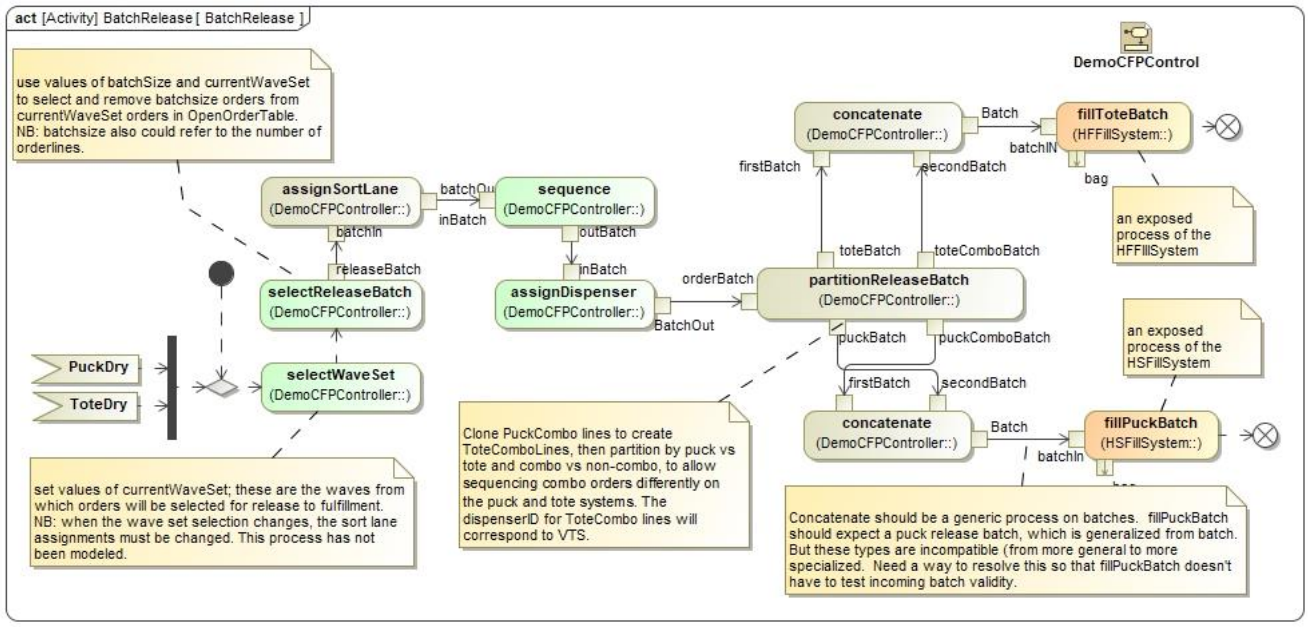

Figure 6-7 *DemoCFP* Batch Release to Fulfillment

The specific control decisions and control processes for *DemoCFP* that are used in Figures 6-6 through 6-8 are identified in Figure 6-8. These decisions align with the five decision types defined in the DELS specification:

- AdmitNewOrder: admission
- AssignDispenser: resource assignment
- SelectReleaseBatch: sequencing
- SelectWaveSet: sequencing
- Sequence: sequencing
- SortLaneAssignment: resource assignment

It is interesting to note that the process of partitioning the release batch, and concatenating the partitions differently for the puck line and tote line also is a type of sequencing, but it is a completely mechanical operation on the incoming order batch, based only on the order type.

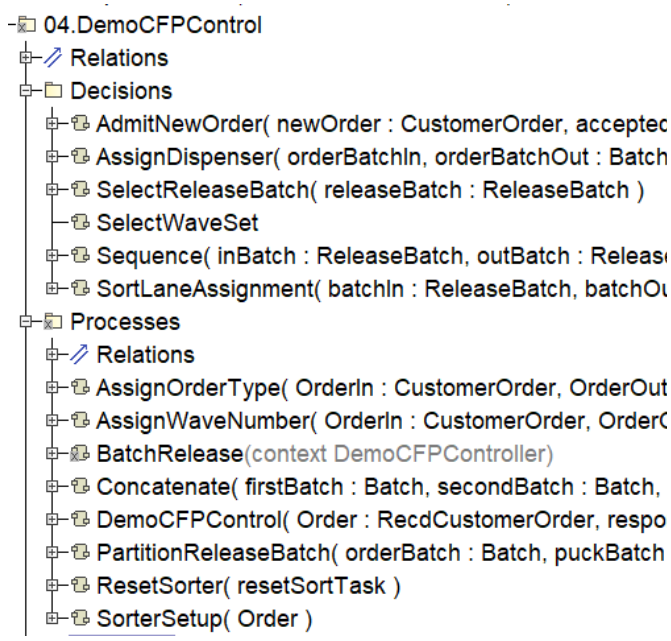

Figure 6-8 *DemoCFP* Control Decisions and Control Processes

In the DELS framework, *task* is the authorization to execute a *process* for a specific product and is identified as a distinct object. In the *DemoCFP*, task takes two forms: (1) a parameter in a call behavior action; and (2) a signal. The "PuckDry" and "ToteDry" signals inform *BatchRelease* to process and release a new batch of orders. Every other process has a set of input and output parameters, and those parameters constitute the information that would be conveyed in an explicit task. The invocation of the behavior, along with the parameter set is the authorization for executing the associated process. In this sense, the task definitions are simply the specification of the parameters themselves, which are summarized in Figure 6-9. Figure 6-10 shows how the various tasks are related to one another and to the generic tasks identified in the *CFP_Abstractions* package.

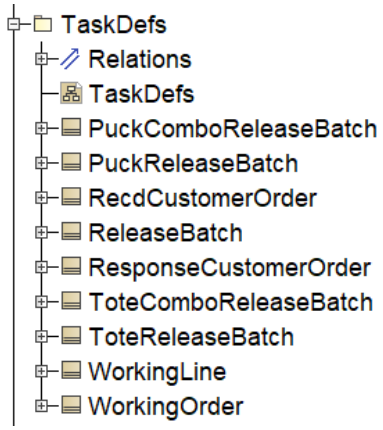

Figure 6-9 Task Definitions for DemoCFP Controller

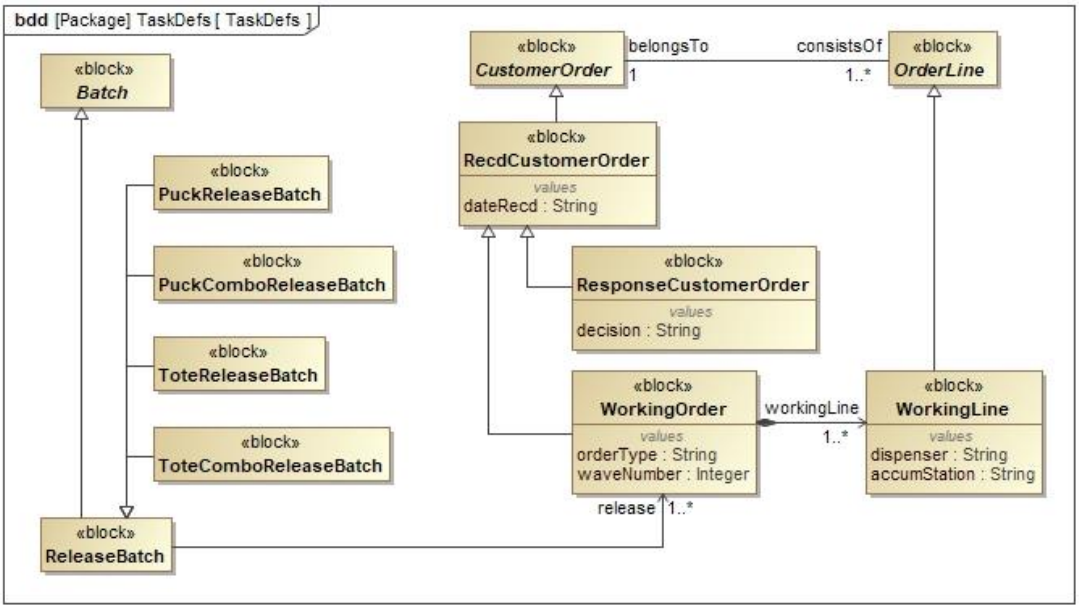

Figure 6-10 Structure of *DemoCFP* TaskDefs

The structure of the *DemoCFPController* is displayed in Figure 6-11. Note that *DemoCFPController* has as part properties several data objects. The data objects, *NDCLocnTable* and *WaveTable* represent information that is the result of planning processes, namely the assignment of NDCs to dispense locations and the assignment of stores to sort waves, respectively. *OpenOrderTable* is the set of received and admitted customer orders that are available to be released for fulfillment. The value property *batchSize* is assumed here to be a constant, also determined by a planning decision (although in a more advanced controller it might be an operational decision). The *currentWaveSet* is the result of a *DemoCFPController* decision and specifies the waves from which orders will be chosen from the *OpenOrderTable* for release to fulfillment.

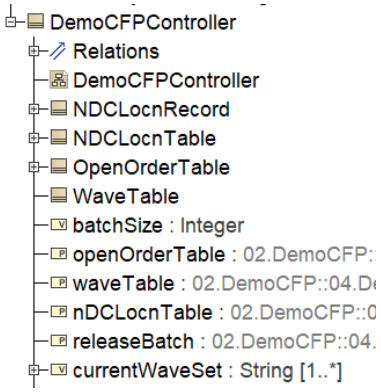

Figure 6-11 *DemoCFP* Controller

This completes the modeling of the *DemoCFP*. The following sections will take a similar approach to describing the modeling of each of *DemoCFP*'s subsystems.

6.2 *HSFillSystem* Package

In the *HSFillSystem* only order lines for the most frequently ordered drugs are dispensed and the primary function of the system is to dispense drugs into vials and accumulate the vials for merging into a customer order. Vials are transported in pucks to individual workstations for dispensing a single NDC, verifying the NDC and quantity, capping and accumulation. Accumulation is at a bagger workstation for orders that can be filled completely in the *HSFillSystem* and in the *VTS* for combo order lines.

6.2.1 *HSFillSystem* Product

As shown in Figure 6-12, the only products of the *HSFillSystem* are a bagged customer order, deposited on the take-away conveyor to the order-to-store sorting system, or a combo line deposited in the *VTS*.

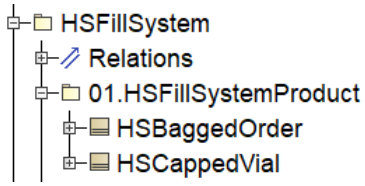

Figure 6-12 *HSFillSystem* Products

6.2.2 *HSFillSystem* Resource

The active resources in *HSFillSystem* are illustrated in Figure 6-13. On the left side of the figure are the resources involved in dispensing pills into vials, verifying the NDC and count, capping vials and accumulating vials in order to bag customer orders. On the right side of the figure are blocks representing elements of two conveyor systems—the puck conveyor that moves vials among the workstations shown on the left, and the take-away conveyor that removes bagged orders.

The vial transfer system, *VTS*, is a reference property rather than a part. Note that the puck conveyor, which is a part property of *HSFillSystem*, provides an interface position at the *VTS*. In a similar fashion, the take-away conveyor, which is not a part property of *HSFillSystem*, provides an interface position to each of the bagger workstations. *VTS* is modeled in more detail in section 6.4.

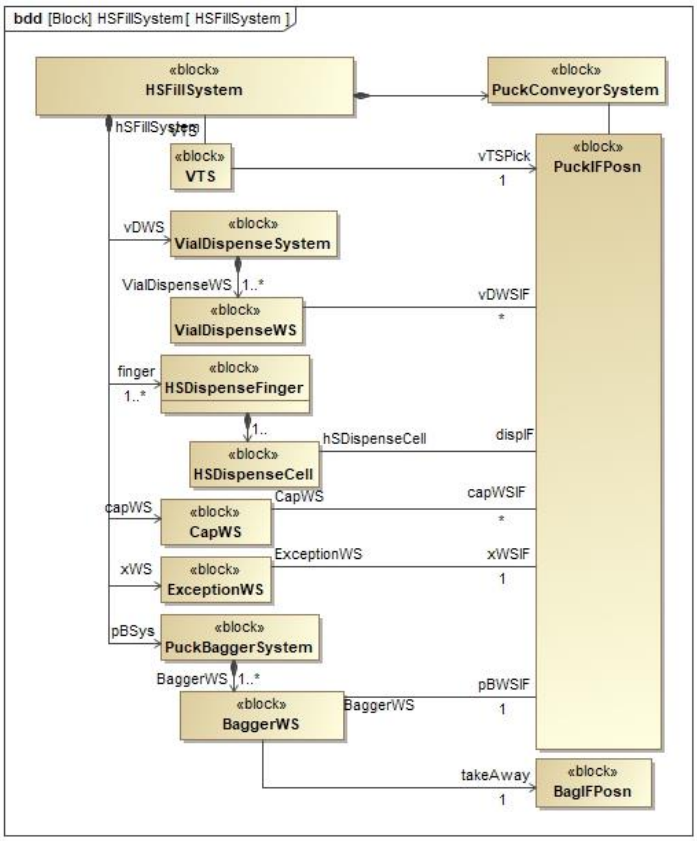

Figure 6-13 *HSFillSystem* Resources

The *VialDispenseSystem* consists of one or more *VialDispenseWS*, or workstations. Each is capable of dispensing and labeling a vial and depositing it in an empty puck. The *VialDispenseSystem* exposes a single callable operation, *dispenseVial* whose method is the process *DispenseVialMarryPuck* . This process calls the *dispenseVial*, *labelVial*, and *insertVial* operations of a *VialDispenseWS* as well as the *selectWS* operation of the *VialDispenseController*.

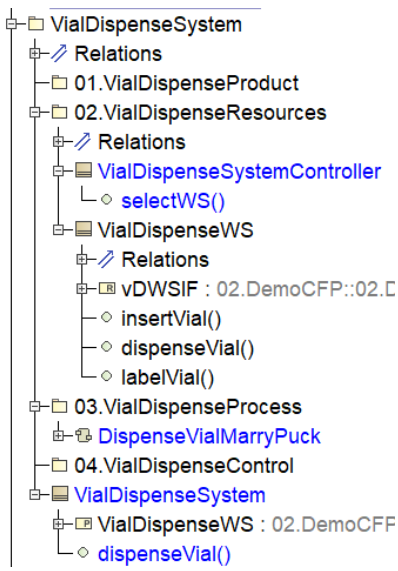

Figure 6-14 *VialDispenseSystem* Model

The *HSDispenseFinger* consists of multiple *HSDispenseCells*, each of which can dispense a single NDC into a vial presented by a puck at its designated *PuckIFPosn*. It also has as part properties an *ImageWS* and a *ScaleWS*. Figure 6-15 shows an example of a *HSDispenseFinger* with eight dispense cells. Note that every part of the finger that performs an operation involving a vial in a puck has as a reference property a corresponding *PuckIFPosn* which is a part property of the associated *FingerSpur* conveyor part. The *HSFillSystem* may have several *HSDispenseFingers*.

In this model there is a single *CapWS* where vials in pucks are capped and a single *ExceptionWS* which resolves errors that may have occurred for a puck at any of the previously visited workstations.

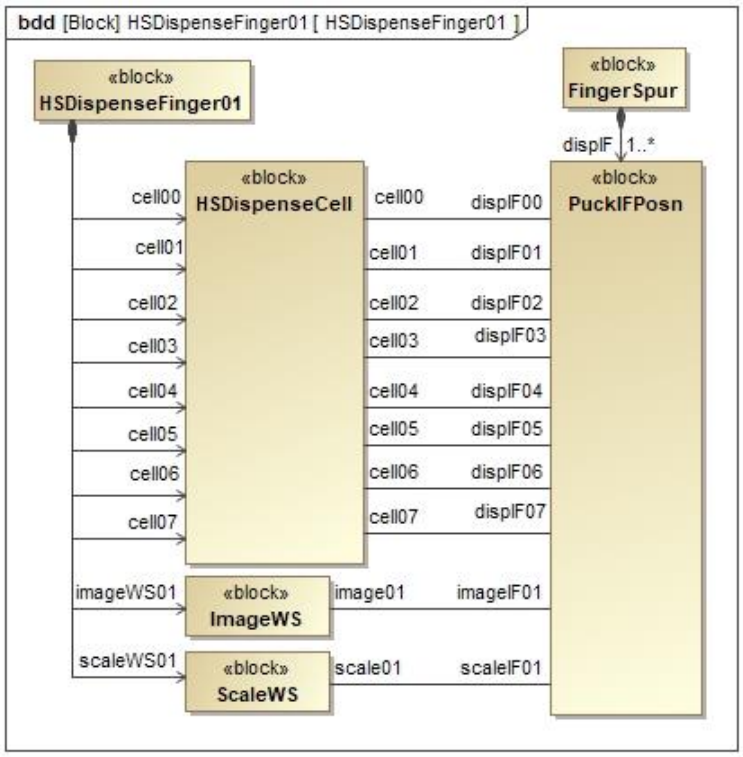

Figure 6-15 *HSDispenseFinger* Example

Finally, the *PuckBaggerSystem* consists of multiple *BaggerWS* where customer orders are collected and when complete are placed in a bag along with all required documentation. Figure 6-16 illustrates a puck bagger system with two bagger stations. A *BaggerWS* has an associated interface position from both the puck conveyor system and the tote conveyor system.

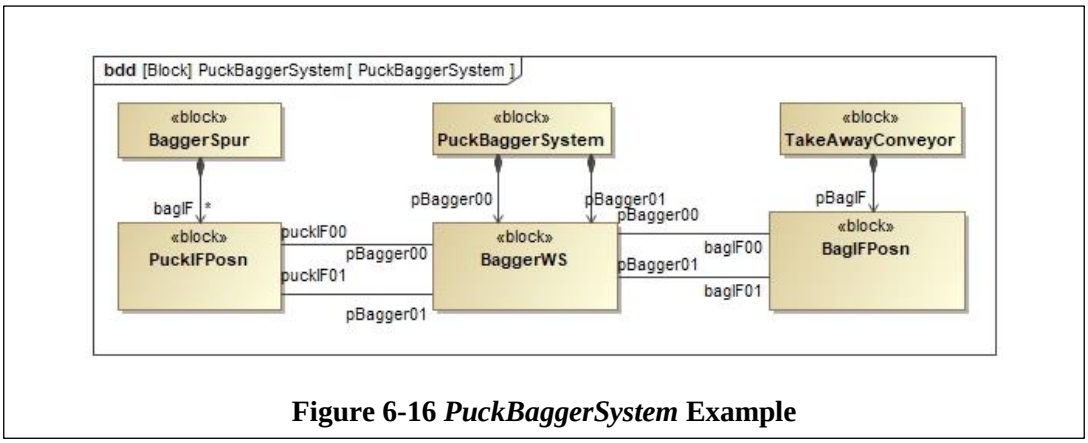

Figure 6-16 *PuckBaggerSystem* Example

The organization of the resources in *HSFillSysem* is summarized in Figure 6-17. For every major component, except *ExceptionsWS* there is a spur conveyor which provides some queuing at the workstation. The *MainPuckLoop* provides overall puck circulation. Both the *VTS* and *TakeAwayConveyor* are reference properties, thus shown with dashed borders.

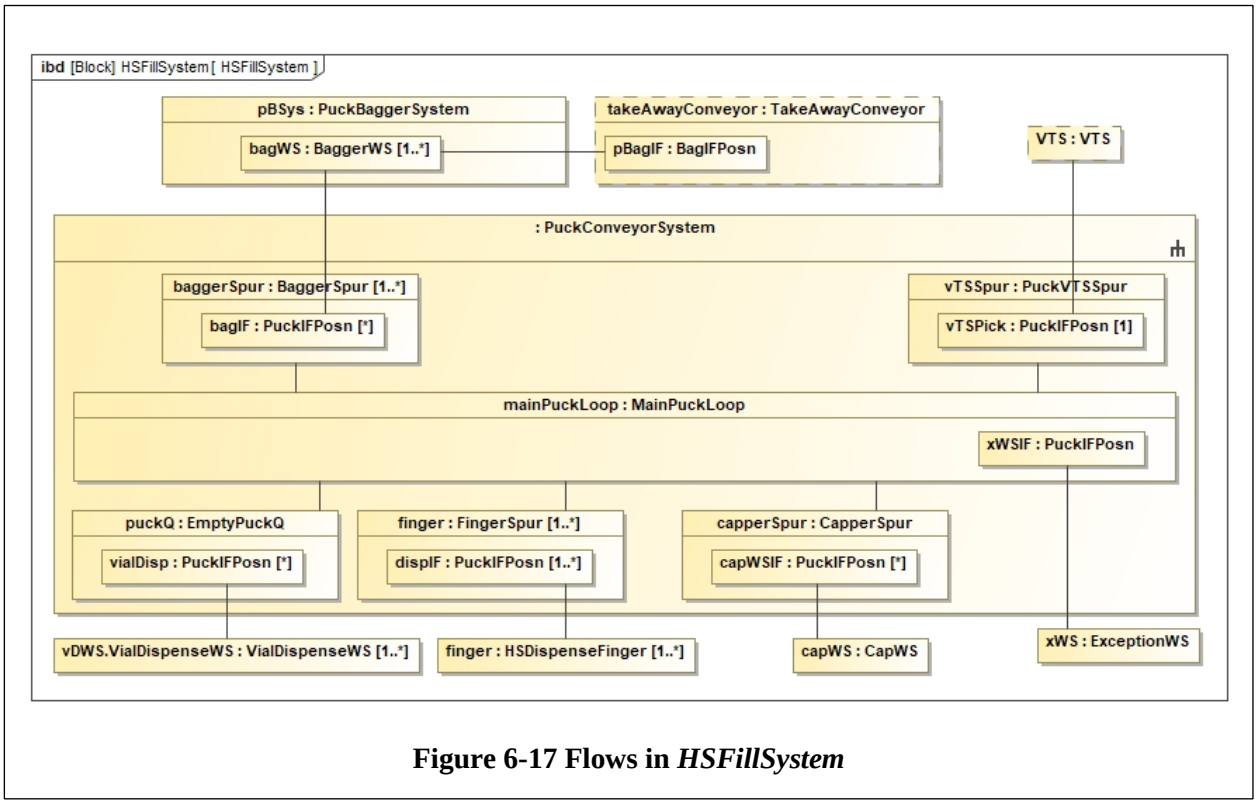

Figure 6-17 Flows in *HSFillSystem*

6.2.3 *HSFillSystem* Process

The *HSFillSystem* has four processes that either are exposed to the *DemoCFP* or directly impact a *HSFillSystem* owned resource. The *PuckFillOrderBatch* process is exposed to the *DemoCFP* as the call operation action *fillPuckBatch* of the *HSFillSystem* and is shown in Figure 6-18.

A batch of augmented orders, created by the *BatchReleaseProcess* of *DemoCFPControl*, is the input to the *PuckFillOrderBatch* process. This batch is added to a data store, *ActiveBatch* which is a part property of the *HSFillSystemController*. All these orders are marked as "open", i.e., they are available to release for fulfillment.

A mechanism is needed to meter pucks into the *HSFillSystem* so that congestion on the conveyor or at heavily used workstations is avoided. The mechanism illustrated in Figure 6-18 is based on the number of pucks currently in process with a vial. When this "work in process" or puck WIP is below a target level, a new order can be released and as its order lines are released, the puck WIP will increase. If there is an open order that has not yet been released, it will be selected and two actions follow: (1) check the puck WIP until it falls below the puck WIP target (because vials are being accumulated either at the *VTS* or at a bagger workstation); and (2) fill the order using the process appropriate to the type of order.

The difference between the two puck order fill processes is that combo order lines are accumulated in the *VTS* and thus do not require bagging, while non-combo lines are accumulated at a bagger station and when complete, must be bagged. Figure 6-20 shows the process for filling a combo order and Figure 6-19 shows the process for non-combo orders.

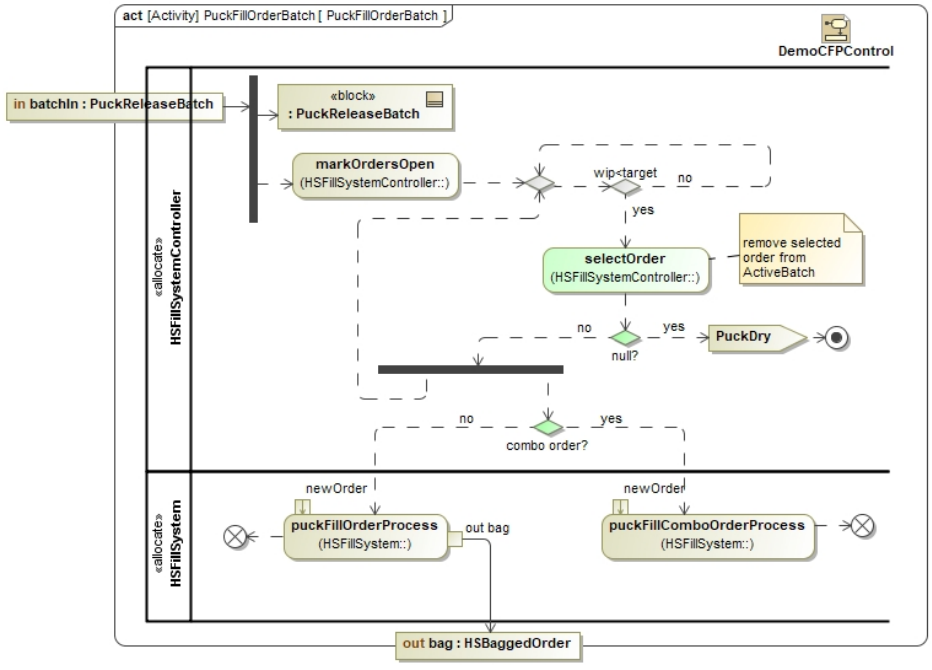

Figure 6-18 Puck Fill Order Batch Process

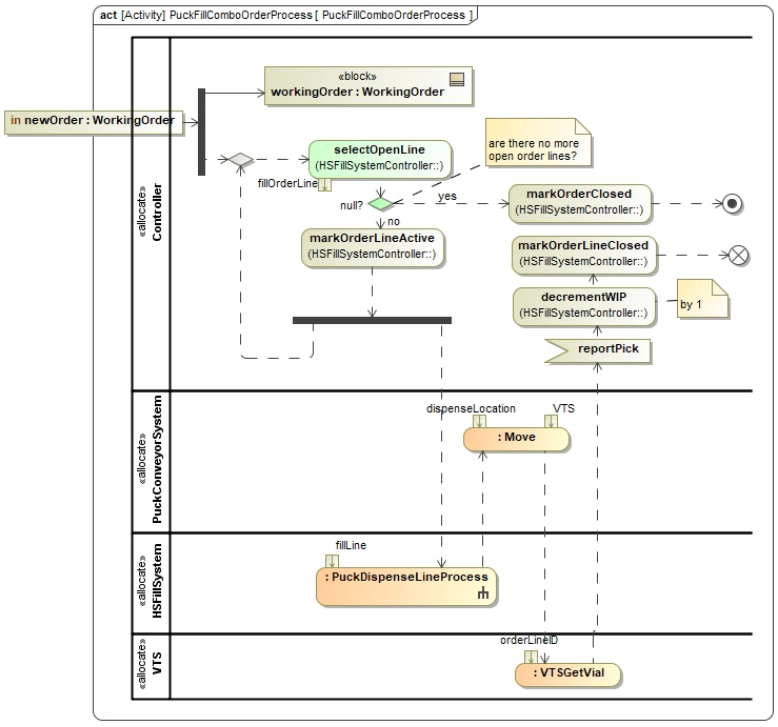

Figure 6-19 Puck Line Fill Combo Order Process

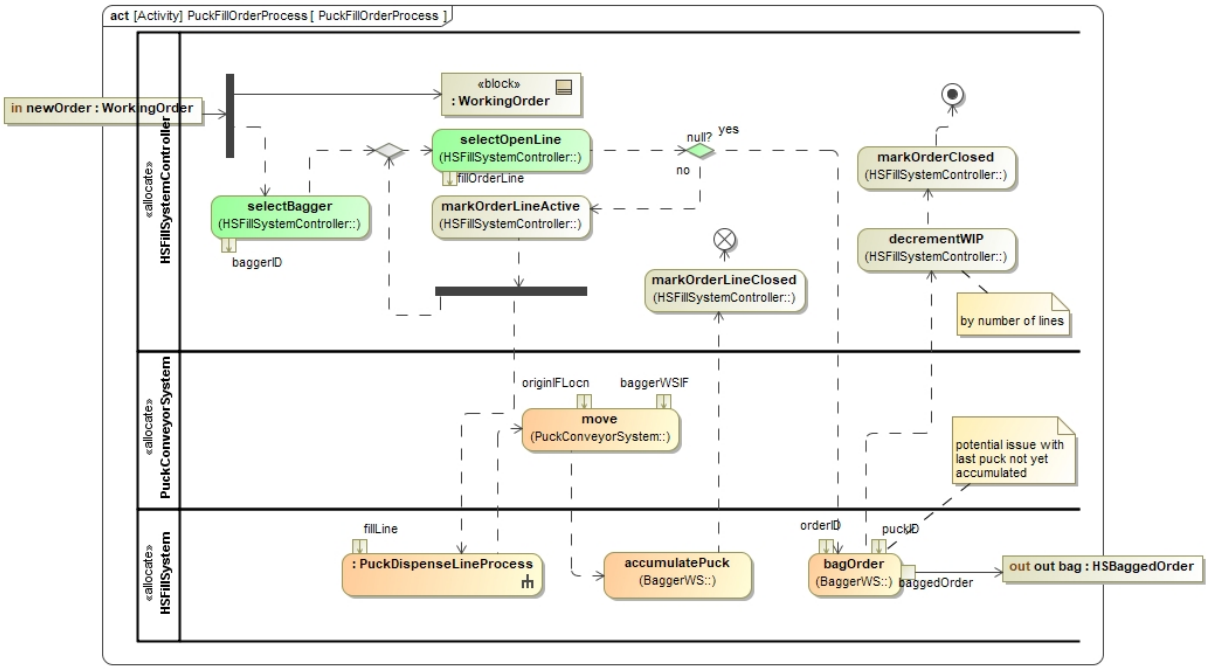

Figure 6-20 Puck Line Fill Non-combo Order Process

Both Figure 6-20 and Figure 6-19 have a call behavior action referring to the *HSFillSystem* operation *puckDispenseLineProcess* whose method is *PuckDispenseLineProcess*. This process is described in Figure 6-21 and involves the *HSFillSystemController*, the puck conveyor and the dispense resources of the *HSFillSystem*. The proper sequencing of process steps is managed through the execution of this process. Note that the process can terminate in two ways: (1) vial is successfully capped; or (2) an exception is resolved. Exceptions occur when there is an error in any of the dispense processes, from vial dispense through capping. For simplicity, it is assumed that resolving an exception results in a capped vial with the correct quantity of the correct NDC. Thus, when this process terminates, the line filled is ready to be accumulated. Where it is to be accumulated is determined by the type of order, as illustrated in Figure 6-18 and Figure 6-20.

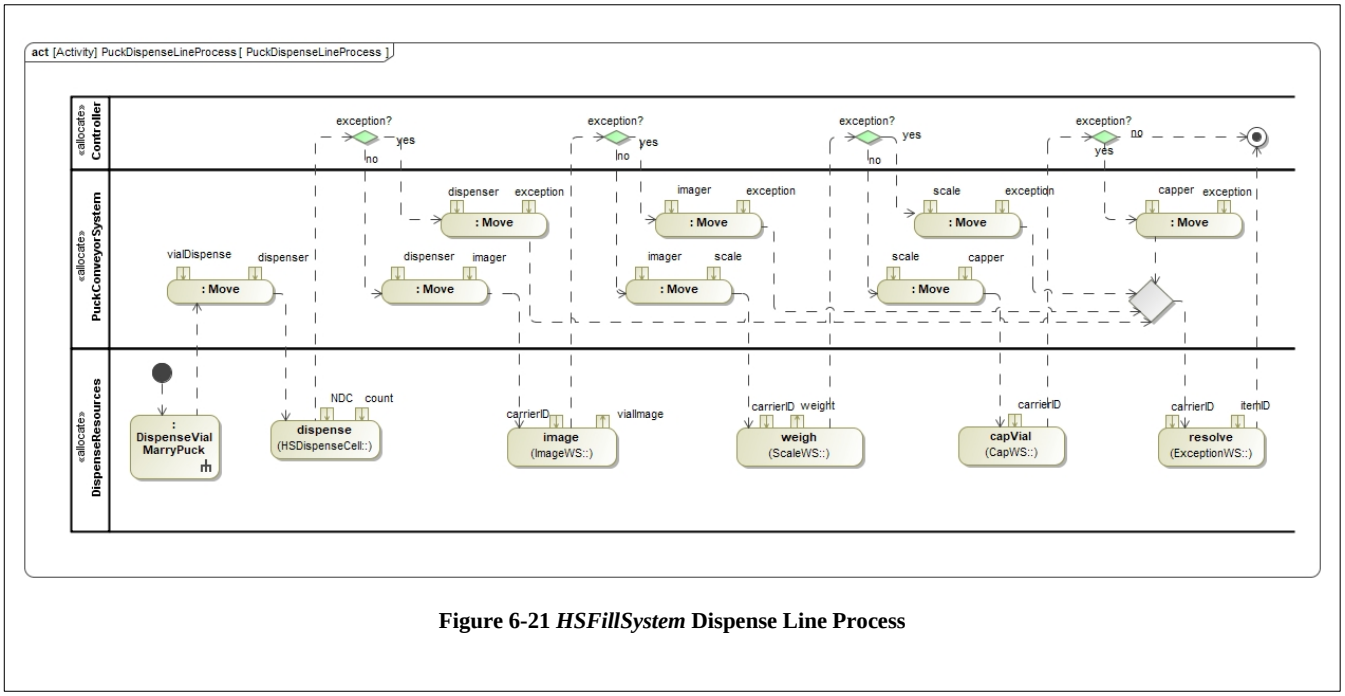

Figure 6-21 *HSFillSystem* Dispense Line Process

6.2.4 *HSFillSystem* Control

As shown in Figure 6-18, the *HSFillSystemController* receives a batch of orders when its *PuckFillOrderProcess* is invoked by the *DemoCFPController* call to the *fillPuckBatch* operation of *HSFillSystem*. In Figure 6-22, the associated decisions, processes and tasks definitions are identified. There are three control decisions listed:

- *HSSelectBagger*: associates a particular bagger workstation with an order; the simplest decision process would be round-robin, although least work or other rules could be used;
- *HSSelectOpenOrder*: from the *PuckReleaseBatch*, select the next order to process; the simplest rule is to use the sequence of orders in the batch;
- *HSSelectOpenLine*: for the open order currently being filled, select which line to fill next; the simplest rule is to use the sequence of lines in the order.

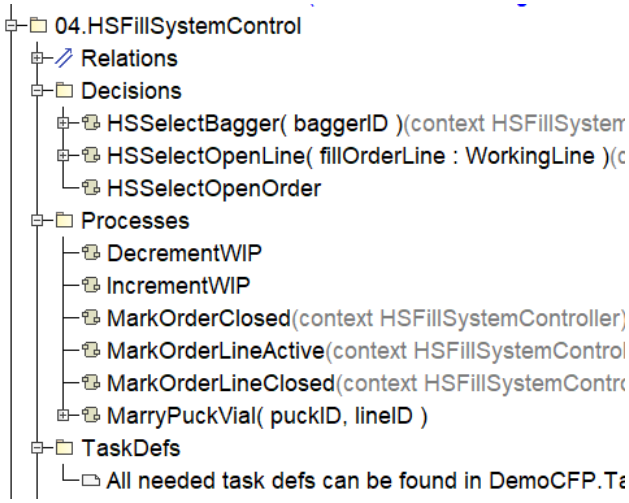

Figure 6-22 *HSFillSystemControl* Decisions, Processes and Task Definitions

Additional control decisions are illustrated in Figure 6-18 through Figure 6-21, where there are simple logical branches for type of order, or presence of an order, or occurrence of an exception event. In each case the decision is a simple branching, based on state. .

The processes listed in Figure 6-22 all are information management processes necessary to support the decisions. The tasks defined for the *DemoCFPController* are sufficient for the *HSFillSystemController*.

6.3 *HFFillSystem* Package

The *HFFillSystem* is capable of dispensing any NDC that is available in the *DemoCFP*. It is comprised of manual workstations, automated dispensers for certain unit-of-use drugs, and robotic workstations employing automated dispensers that are similar to or identical with those used in the *HSFillSystem*. While the *HSFillSystem* may have capability for fewer than 100 NDCs, the *HFFillSystem* may have capability for several thousand NDCs.

6.3.1 *HFFillSystem* Product

The product of the *HFFillSystem* is a bagged customer order, placed on the *TakeAwayConveyor*.

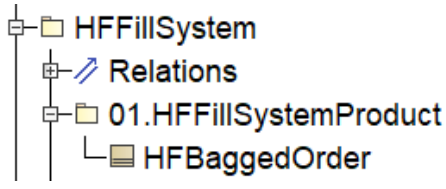

Figure 6-23 *HFFillSystem* Product

6.3.2 *HFFillSystem* Resource

The resource organization for *HFFillSystem* is illustrated in Figure 6-24. The major components are shown on the left side of the diagram and all are part properties except for *VTS* which is a reference property. The *ToteConveyorSystem* is a part property, and it also is a DELS. Its components include the seven spurs shown, and each of them provides an interface position for some dispense resource. Note that all the dispense resources may have multiple instances in a given system configuration and the *ToteBaggerSystem* may have multiple *ToteBagger* workstations.

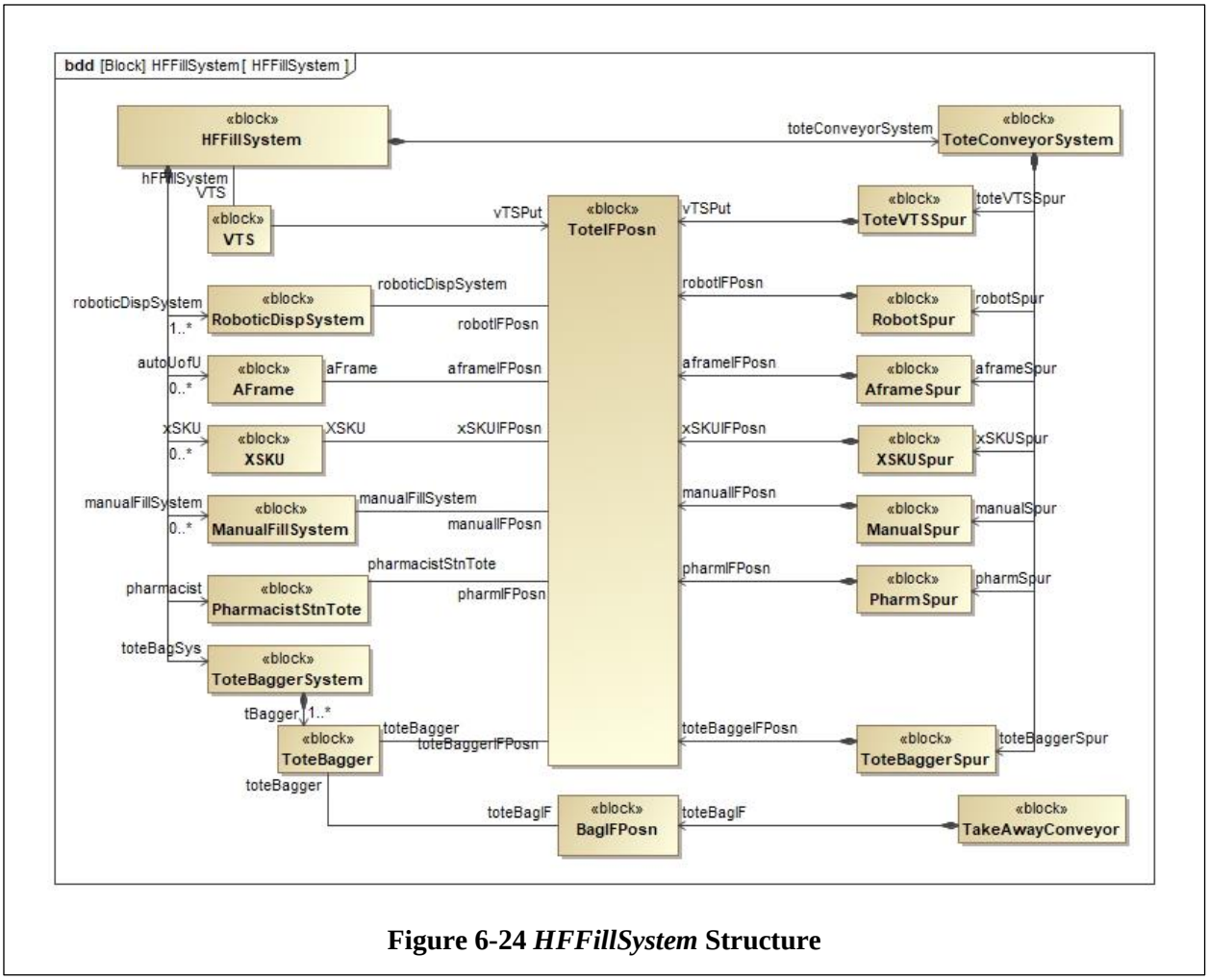

Figure 6-24 *HFFillSystem* Structure

The structure of the *HFFillSystem* is further elaborated in Figure 6-25. As in the *PuckConveyorSystem*, the *ToteConveyorSystem* provides a spur for each major dispense resource and an interface position where the resource can access the tote to deposit a line item.

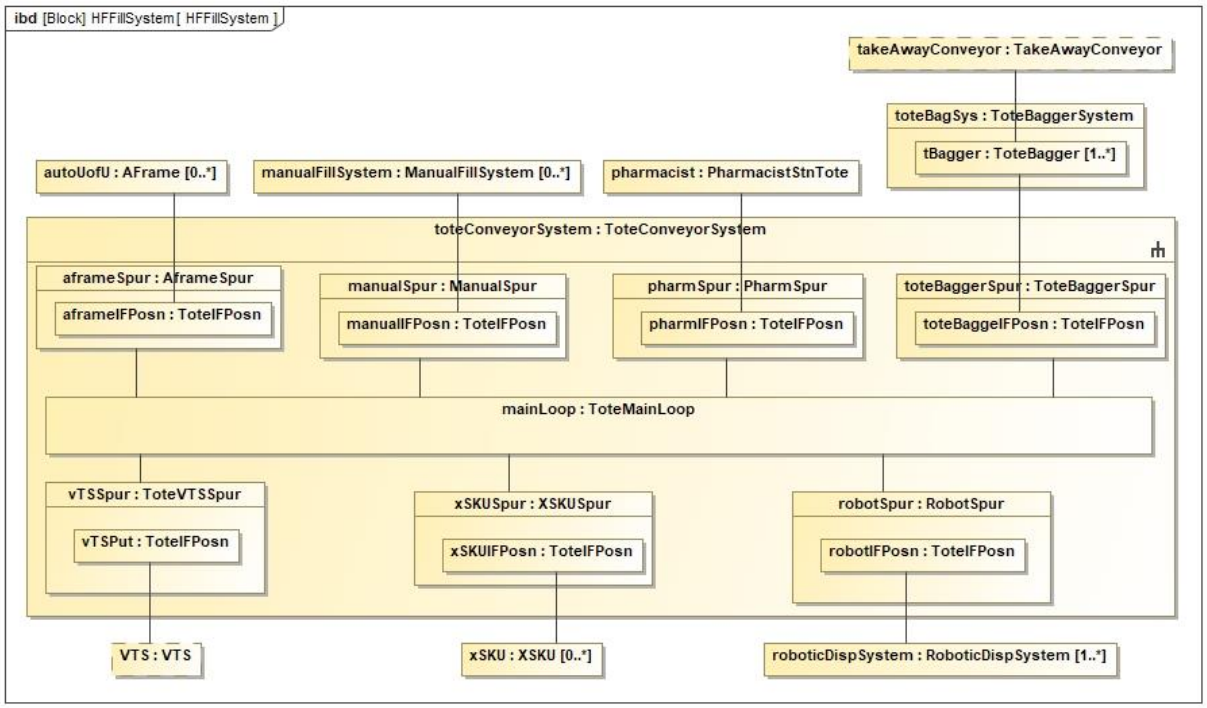

Figure 6-25 Flows in *HFFillSystem*

6.3.3 *HFFillSystem* Process

HFFillSystem has two defined processes. *FillToteBatch* is exposed to *DemoCFP* through calls to the *fillToteBatch* operation of *HFFillSystem* which has *FillToteBatch* as a method. Similarly, *FillToteOrder* is invoked by *FillToteBatch* through a call operation action on *fillOrder* operation of *HFFillSystem* which has *FillToteOrder* as a method.

FillToteBatch is shown in Figure 6-27 and is very similar to *PuckFillOrder Batch*, except that there is only one process for filling a tote order. Because *FillToteOrder* is exposed as an operation of *HFFillSystemController*, the parameter *batchIN* provides the input of the *ToteReleaseBatch* directly to the *HFFillSystemController* where it goes into the datastore for subsequent manimpulation by the controller. The *selectOrder* decision can be quite simple, e.g., preserving the order sequence in the *ToteReleaseBatch*, or it could be more complex, depending on the implementation. When the selected order is "null" (i.e., no more orders in the data store) the controller signals that *HFFillSystem* is ready for another batch.

FillToteBatch invokes the *FillToteOrder* process by a call to *fillOrder*, an operation of *HFFillSystem* which has *FillToteOrder* as its method. As shown in Figure 6-26, the parameter *ToteOrder* provides the order to be filled, *WorkingOrder* directly to the *HFFillSystemController*, which first assigns it to a specific tote via the *marryToteOrder* operation which has the process *Marry* as its method. Once the order is married to a tote, the value of *wIP* is incremented and the tote is routed to a series of stations corresonding to the NDCs identified in the order lines. Once the "next line" is "null", a bagger is selected and the routed to the bagger. A simple method for choosing the bagger station is round-robin, but other rules could be implemented.

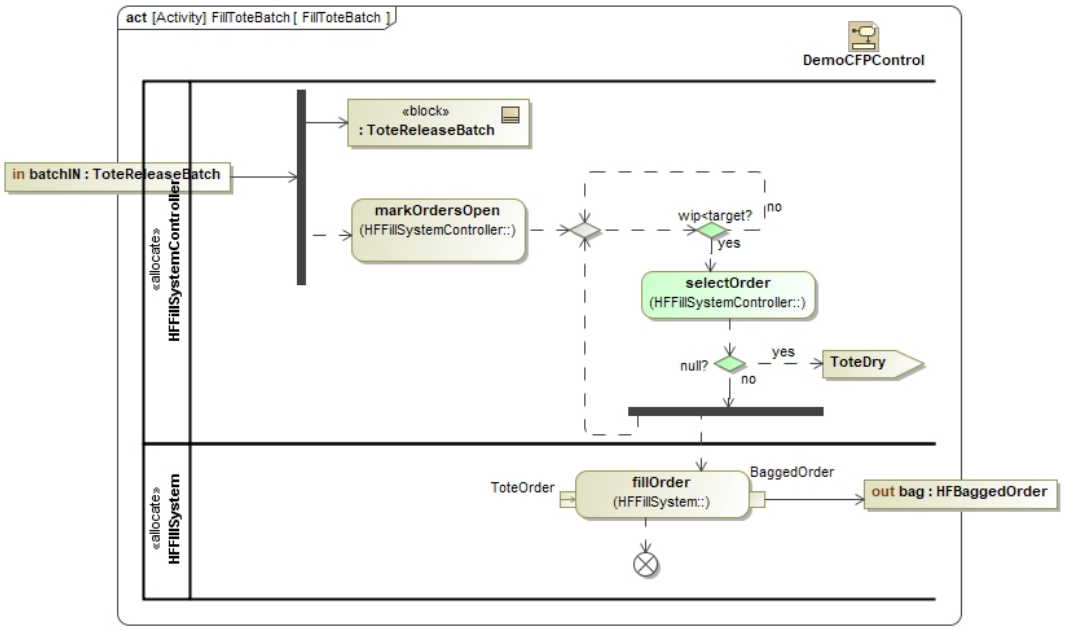

Figure 6-27 *FillToteBatch* Process

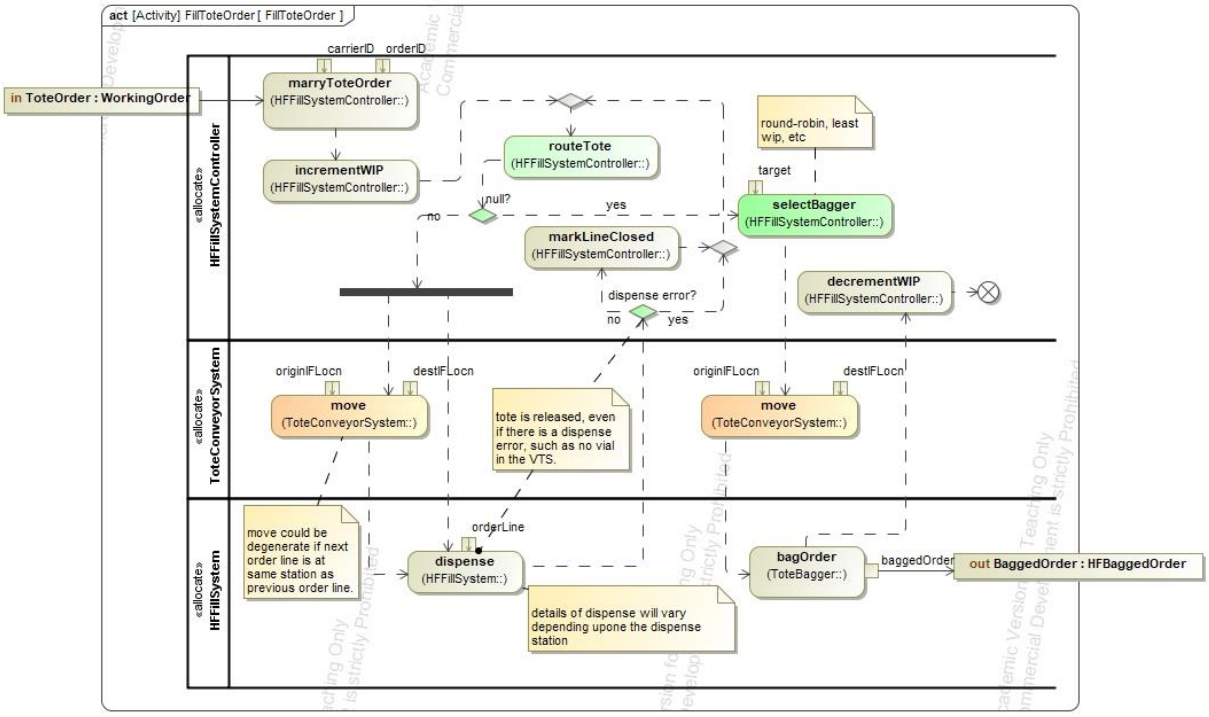

Figure 6-26 *FillToteOrder* Process

6.3.4 *HFFillSystem* Control

HFFillSystemController makes three decisions that are modeled as activities:

- *HFSelectBagger* simply determines which of the available baggers on the tote line will bag the order contained in a tote. The rule could be as simple as round robin, or could be more complex, e.g., bagger with fewest queued totes.
- *HFSelectOrder* determines the next order to remove from the *ToteReleaseBatch*, which could be as simple as using the sequence in the *ToteReleaseBatch*.
- *HFSelectWS* determines the next order line to fill and the corresponding workstation; this could be as simple as taking the next order line in sequence and looking up the workstation for the corresponding NDC, or it could determine the line whose corresponding workstation is closest to the current location of the tote.

As with the *HSFillSystemController* there are a number of control decisions that are simple binary choices based on the state of a query to a data store, WIP level relative to WIP target, or occurrence of a dispense error. Similarly, the *HSFillSystemController* has

6.4 *VTS* Package

The *VTS* plays a key role in filling combo orders. For the *VTS* to function properly, and for the *DemoCFP* to be able to successfully fill combo orders, the design of the *VTS* must be integrated with the design of the rest of the *DemoCFP*.

6.4.1 *VTS* Product

The only product of the *VTS* is a vail corresponding to an order line, deposited into a tote corresponding to the order containing the order line.

6.4.2 *VTS* Resource

The VTS has two main components, a robot and a vial store. The robot in this application is simply an equipment (with its own L2 controller) that executes vial transfers on command. The transfers are from a puck to the vial store, or from the vial store to a tote. The vial store is simply a set of slots where a vial can be deposited and later retrieved by the robot.

The components of the VTS are summarized in Figure 6-28. The *vTSPick* and *vTSPut* reference properties are for the conveyor interface locations where the robot, respectively, can pick a vial from a puck and put a vial to a tote. Because these two locations never change, the only "variable" in the robot moves is the slot to which the vial is stored after being picked from the puck, or from which it is retrieved prior to being put in the tote.

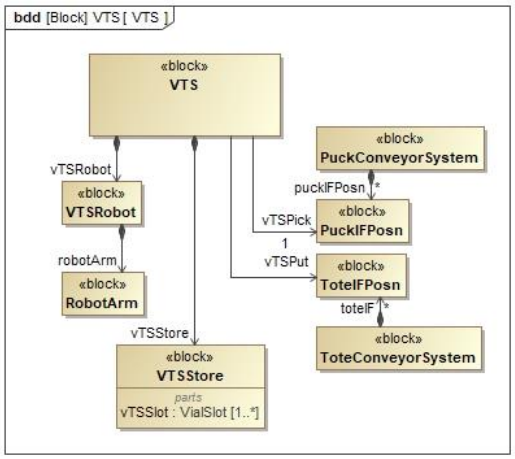

Figure 6-28 VTS Structure

The internal structure of the VTS is shown in Figure 6-29. Included in this figure is the VTS controller and the robot controller. The VTS is a DELS, and has an ISA-95 Level 3 controller, because it must determine which of the conveyor interfaces will be served next whenever the robot completes a move, as well as which vial slot will be involved. The robot, in contrast, only executes the moves determined by the VTS controller, so it is an equipment, and has an ISA-95 Level 2 controller.

An important observation regarding the structure of *VTS* is that at most one puck can occupy *vTSPick* and at most one tote can occupy *vTSPut*.

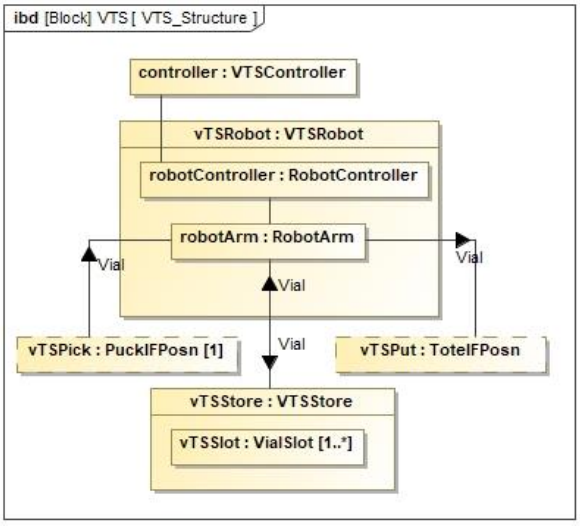

Figure 6-29 VTS Internal Structure

6.4.3 *VTS* Process

VTS has two operational processes, *VTSGetVial* and *VTSPutVial*. *VTSGetVial* is exposed to *HSFillSystem* via the *vTSGetVial* operation of the *VTS* which has *VTSGetVial* as its method. Similarly, *VTSPutVial* is exposed to *HFFillSystem* via the *vTSPutVial* operation of the *VTS* which has *VTSPutVial* as its method.

The design of the *VTS* is predicated upon a key assumption about the way *HSFillSystem* and *HFFilSystem* interact with *VTS*. In particular, *vTSGetVial* is not called until the puck conveyor has delivered the target puck to *vTSPick*, and likewise, *vTSPutVial* is not called until the tote conveyor has delivered the target tote to *vTSPut*.

6.4.4 *VTS* Control

When the *VTS* operation *vTSGetVial* is called by *HSFillSystem* the corresponding *FillOrderLine* is added to *PickTaskTable* a part property of the *VTSController*. Similarly, when the *VTS* operation *vTSPutVial* is called by *HFFillSystem* the corresponding *FillOrderLine* is added to *PutTaskTable*, a part property of the *VTSController*.

The controller process *VTS_Control* is illustrated in Figure 6-30.

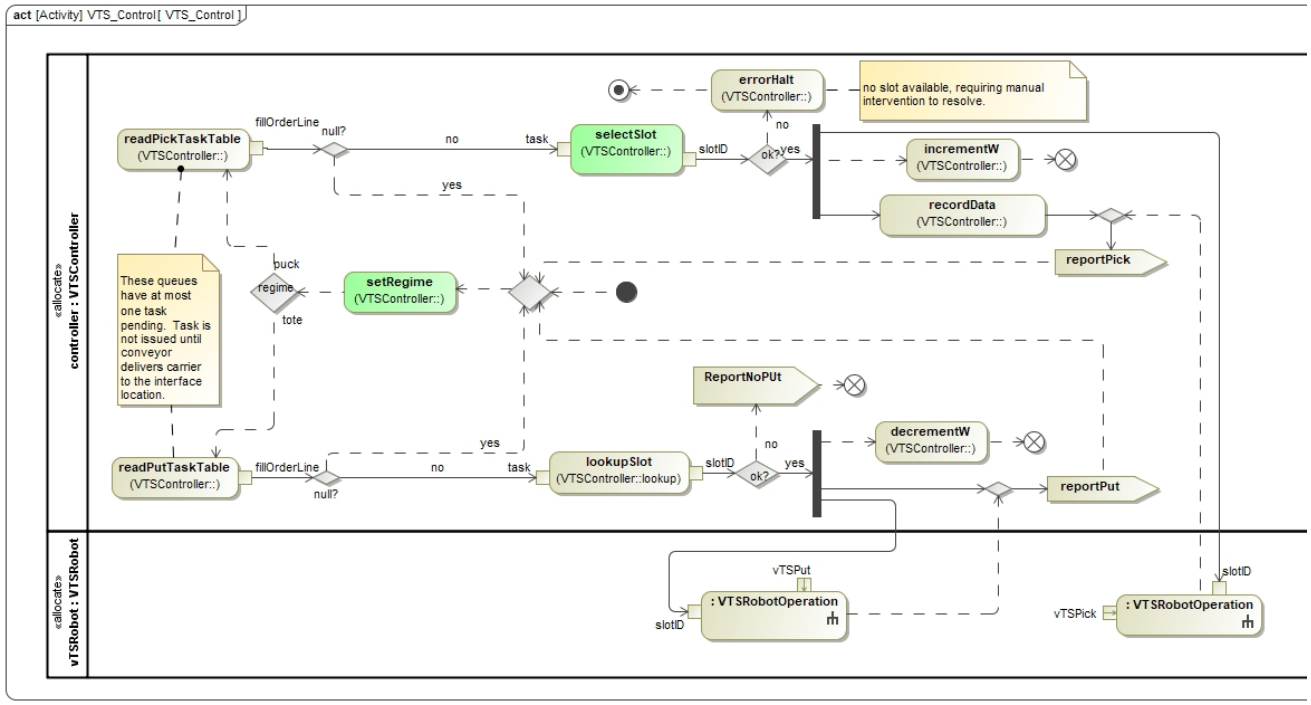

Figure 6-30 *VTS* Control Process

Once the control process is initiated it runs continuously. The *VTS* operates in one of two regimes, serving either the puck conveyor or the tote conveyor. In either regime, it focuses on the corresponding interface location until a decision is made to change regime. To illustrate, at the beginning of the day, since combo orders are run first on *HSFillSystem*, it would make sense for the *VTS* to focus on picking vials from the puck conveyor. At some point, it would make sense for the *VTS* to switch regime and begin to put vials in to totes on *HFFillSystem*. The *setRegime* operation of the *VTSController* has as its method the *SetRegime* decision process. There are a number of ways to make the decision in *SetRegime*, for example, alternating, or first-come-first-served. One might even contemplate a *SetRegime* process that takes into account the numbers of combo orders in a batch, or currently active plus not yet released, etc.

In a similar fashion, the *selectSlot* operation has as its method the *SelectSlot* decision activity. The slot selection might attempt to minimize robot travel, for example.

The other decision processes for *VTSController* essentially are administrative functions in support of either decision making or the execution of vial transfers.

Note that the design decision to augment the *FillOrder* and *FillOrderLine* with additional information makes it possible for the *VTS* to control itself, based on nothing more than the tasks specified in calls to *vTSGetVial* and *vTSPutVial*. It knows the order line ID for a vail being retrieved from a puck, and it decides the slot in the *VialStore* and can associate the order line ID with that slot. When there is a subsequent call to retrieve a vial, that call specifies the *FillOrderLine* which allows *VTSController* to match it with the correct vial storage slot.

6.5 *SortSystem* Package

The *SortSystem* plays a key role in *DemoCFP* by assembling the *StoreTotes* containing customer orders for a particular store.

6.5.1 *SortSystem* Product

The product of the *SortationSystem* is the sealed store totes.

6.5.2 *SortSystem* Resource

The major components of the *SortSystem* are shown in Figure 6-3, and the flows are shown in Figure 6-31. As bagged orders reach the *SortSystem*, they are scanned to identify the bagID and thus the corresponding

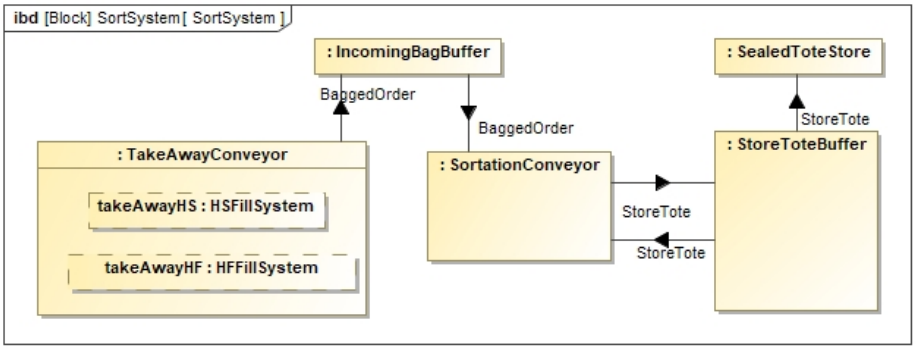

Figure 6-31 Flows in the *Sortsystem*

store ID, then diverted to the appropriate sort lane. Because the *SortationConveyor* may have some lanes re-assigned during the day, partially filled *StoreTote*s may be placed temporarily in the *StoreToteBuffer* and then returned to the *SortationConveyor* when a subsequent re-assignment gives their store a sort lane. At the end of the day, *StoreTote*s are sealed for shipment to the corresponding stores.

6.5.3 *SortSystem* Process

The fundamental process is *Sort* as shown in Figure 6-32. Streams of bagged orders enter the *TakeAwayConveyor* from the *HSFillSystem* and the *HFFillSystem*, and are conveyed to a scanner that reads the bag ID, Based on the bag ID, the *SortationController* determines the assigned sort lane, and then the *SortationConveyor* conveys the bag to its designated sort lane and diverts it.

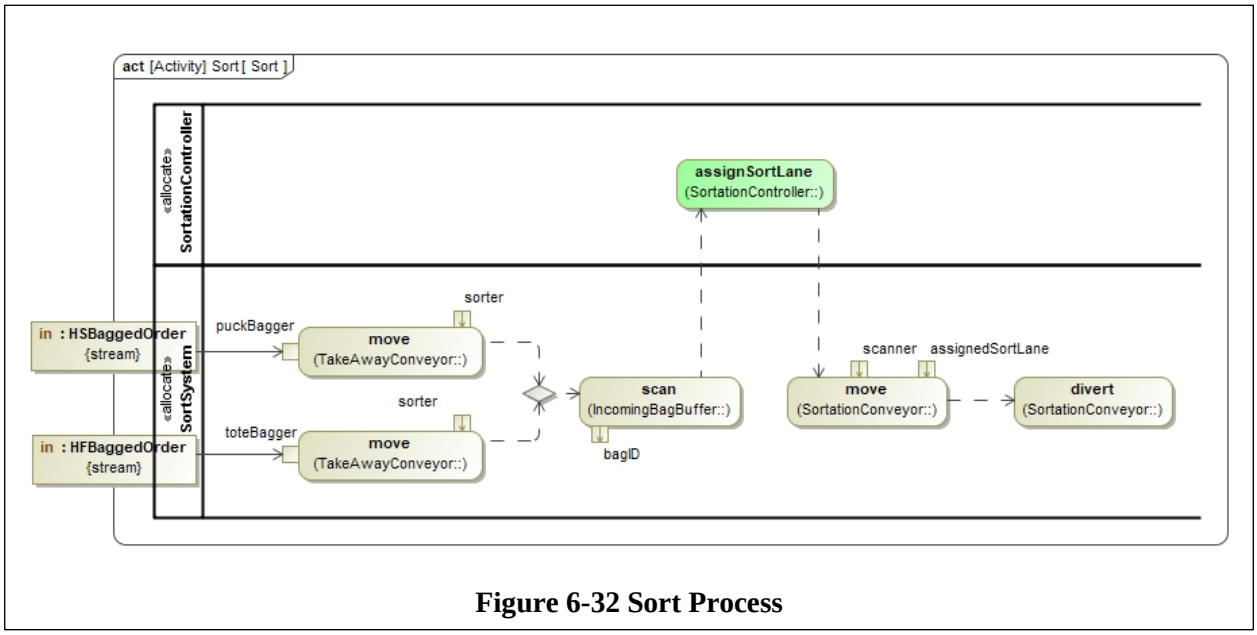

Figure 6-32 Sort Process

6.5.4 *SortSystem* Control

The only local control decision made in the *SortSystem* is the assignment of an incoming bagged order to a specific sort lane. This is determined by the setup of the *SortSystem* which is determined by the *DemoCFPController*. The model presented here does not address this configuration management issue.

6.6 Modeling Summary

A careful reading of this chapter will reveal some modeling practices that have been followed. As pointed out earlier, there currently are no significant SysML-based system models of production systems of a meaningful scale, so there are no "best practices" to follow. The practices followed here may be good, but there may be better practices. Nevertheless, here is a summary of the most significant modeling practices followed here:

- An active resource that has capabilities, modeled as activities, exposes its capabilities through its operations, which can become call operation actions in the behavior of owning or referencing active resources. The signature for these operations is defined in the activity which is the method of the operation. This approach allows the activity model to be modified in any way that does not change its signature, without impacting the called operation actions, which may appear in many places in the system model.

- A corollary is that operations should have methods which are activities. The activity referenced in an operation can be updated, improved, or replaced without changing the way the owning resource's capabilities are accessed.
- An active resource is defined in a package, where the block representing it specifies its parts, references, values and operations (and maybe state) along with a bdd and ibd to understand the organization and interactions of its owned resources. In addition, a good practice is to have owned packages that define:
    - its context and requirements, i.e., what products/services it provides to its users and its capacities (execution rate) for each product/service
    - its owned active resources; note that if these resources also have owned parts, then they will require their own set of defining packages. An exception can be made for "leaf" active resources, where the creation of a package of packages may be less understandable than simply adding all the model elements to the block representing the active resource.
    - its processes, or how its capabilities are realized. These processes define the "signature" of the operation used for invoking the capability
    - Its controls, including the controller with all its value, part and reference properties, and operations, along with owned packages addressing the decision processes (modeled as activities), other control processes (modeled as activities) and task definitions. The latter correspond to the parameters required in any call operation action of a control decision or other control process.
- Many of the process models presented in this chapter use swim lanes. While swim lanes may make it look like actions are being performed by resources, in fact, the "resource" swim lanes often model the call behavior action that invokes the behavior of the resource.
- Finally, the controller is a part property of a DELS or equipment. It does not have a co-equal status, rather it is a co-equal to any of the owned resources of the DELS or equipment.

7 Simulating the DemoCFP

The purpose that drives the development of a model like the one presented in Chapter 6 is to support decision making in both system design and system operation. The SysML model itself primarily captures decisions about resource structure and behavior and about control architecture and decision-making. The focus in this report is on smart operations management, and the appropriate mechanism for evaluating and improving a proposed Level 3 control system is discrete event simulation (DES). This chapter describes a DES model based on the DemoCFP SysML model presented in Chapter 6 and intended to support experimentation with alternative operational control policy parameters and decision algorithms. This is not a detailed documentation of the simulation model, but rather a description of how it represents the DemoCFP, some initial experimental results, and issues that arise id developing such a DES model.

7.1 The Simulation Platform

There are many different DES platforms that might have been used, and each offers advantages and disadvantages. The chosen platform is Simevents™ and MATLAB™ from MATHWORKS. Simevents provides a fairly typical visual editor for constructing the part of the simulation model representing the flow of entities through a system of resources. Like most DES platforms, Simevents supports a variety of queue dispatch disciplines and routing switches but has limited capability for tracking system state or modeling complex control decisions. What Simevents does provide is a mechanism to "escape" from Simevents into MATLAB for computations that are difficult or impossible to implement directly in Simevents, such as maintaining a state database or executing complex decision logic. The escape mechanism is used extensively in the simulated DemoCFP as it allows the maintenance of system state information and decisions that are more than simple queue dispatching.

Examples of state information include the orders that have been accepted but not released in a batch and the orders that have been released and are being processed in the fulfillment systems. This kind of information simply cannot be represented using standard simulation modeling constructs. A number of decisions in the DemoCFP are very different from queue dispatching, including the decision to switch waves, the batching decisions, and the tote routing decision.

7.2 What is Simulated?

The DemoCFP simulation model discussed below corresponds to the left hand side of the system illustrated in Figure 6-6 and the control decisions illustrated in Figure 6-6, i.e., it includes the accept/reject decision for incoming orders, and all the fulfillment operations up to the point where a bagged customer order is placed on the take-away conveyor to the sorting process. Sorting and the accumulation of customer orders in store-specific totes is not included.

7.3 Overview of the Simulation Model

The simulated order stream is based on data for two months of operations for a specific CFP. The data is in the form of a table in which each record corresponds to an order line, with an order ID, a line ID, an NDC, the arrival time of the order and other attributes. The simulation model uses this table to create simulation entities corresponding to each order line and releases these order lines to the simulated DemoCFP at the simulated time of the actual order arrivals.

The simulation model uses MATLAB function blocks to gain access to the capabilities of generic MATLAB for storing and manipulating data and for implementing control decision algorithms. Figure 7-1 illustrates this for the accept/reject decision. The block labelled "Attribute Function" accesses a MATLAB function that compares the NDC in the order line entity to the NDC's in a preloaded NDCMaster and routes the entity to the appropriate port, i.e., either rejected or admitted.

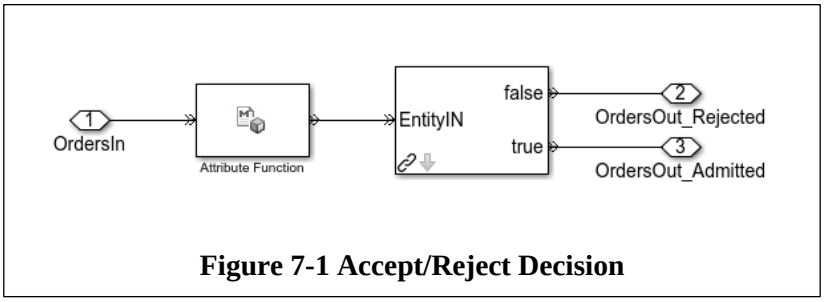

Figure 7-1 Accept/Reject Decision

Admitted order entities are assigned a wave ID attribute, corresponding to the originating store, and then the attributes of the entities are used to create entries in the pending order table, as illustrated in Figure 7-2. Note that MATLAB functions are used both to assign the wave ID to the entity and to create the entries in the pending order table, which is persistent in the MATLAB workspace. Once the entry in the pending order table is created, the store order entity is discarded.

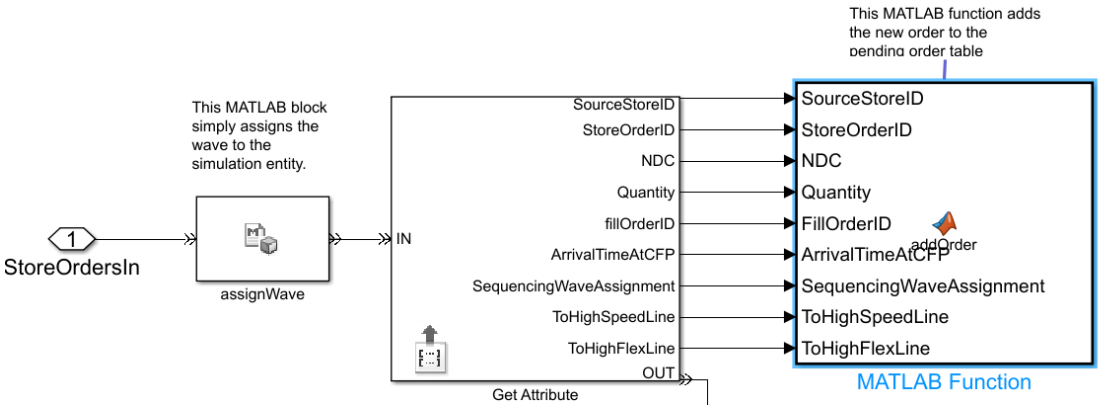

Figure 7-2 Creating the Pending Order Table

The structure and flow of the DemoCFP simulation model closely follows the system model presented in Chapter 6, although there are some differences. The basic structure of the simulation is shown in Figure 7-3 below. There are two controllers, shown on the left side of the figure. The Batch Controller releases batches of customer orders to fulfillment. The Fulfillment Controller releases individual orders and order lines to the high flex and high speed fulfillment subsystems, respectively.

The Batch Controller in Figure 7-3 corresponds to the BatchRelease activity shown in Figure 6-7. The implementation of the Batch Controller is somewhat different from Figure 6-7, however. The approach taken in Figure 6-7 is to release separate order batches to the puck and tote systems, by duplicating the combo lines in the tote orders, and also sequencing the released orders so that combo lines in the puck system are done first and combo lines in the tote system are done last. The release rate is controlled by

limiting the number of pucks and totes with assigned order lines or orders. The implementation of the Batch Controller in Figure 7-3 simply selects the first *b* order lines from the pending orders table, plus any remaining order lines to complete an order. The batch is released to the Fulfillment Controller, where sequencing decisions are made. The resource assignment decision (in the case of multiple feeders with the same NDC) are made in the simulation model Batch Controller as they are in the system model BatchRelease.

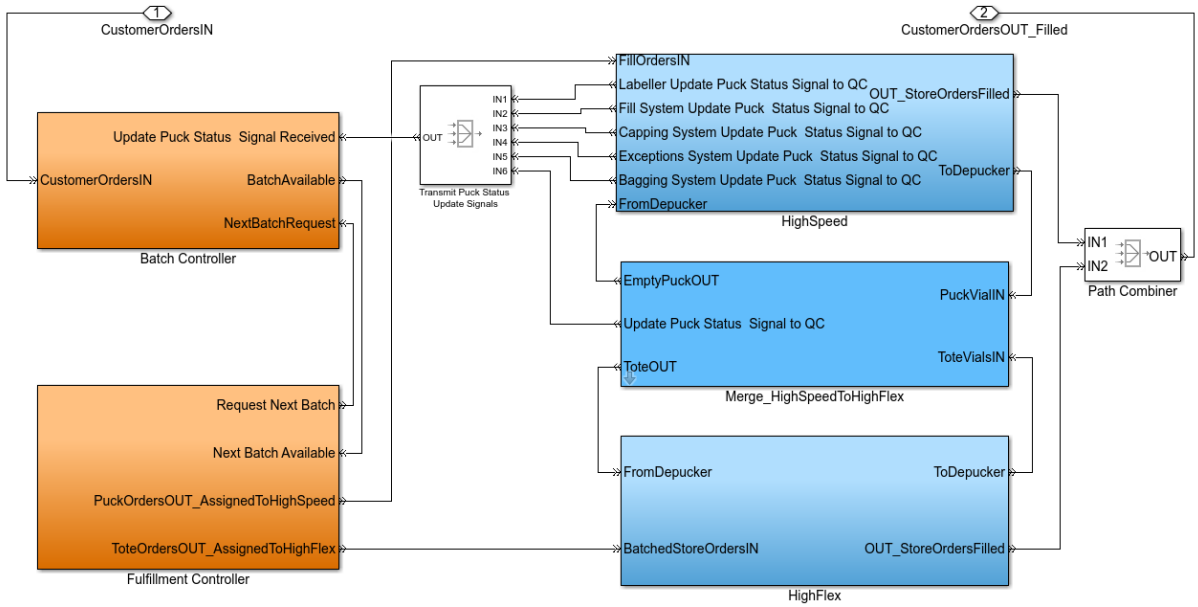

Figure 7-3 Structure of the DemoCFP Simulation

Finally, before releasing the batch, the Batch Controller assigns the feeder for each of the lines in the batch, based on the assignment of NDC to feeders. If the specified NDC for a puck line is available in multiple feeders, the feeder with the least pending work is selected.

The Fulfillment Controller takes the batch of pending orders and releases them in FIFO sequence. The tote system lines are combined into a single simulation entity and released to the HighFlex block and the puck system lines are converted, one-by-one into simulation entities and released to the HighSpeed block. In effect, all the lines in the batch of pending orders are released at the same time, and the two fulfillment systems manage the assignment of lines to pucks and totes.

The next three sections address, in turn, the HighSpeed, HighFlex and Merge_HighSpeedToHighFlex models.

7.3.1 HighSpeed Model

The HighSpeed block explicitly represents the fulfillment resources identified in Chapter 6, using library blocks developed specifically for modeling central fill pharmacy resources. The library for HighSpeed blocks is shown in Figure 7-4. Note that many of these blocks are subsystem masks, meaning they contain a subsystem model and specify parameters that can be set for the subsystem.

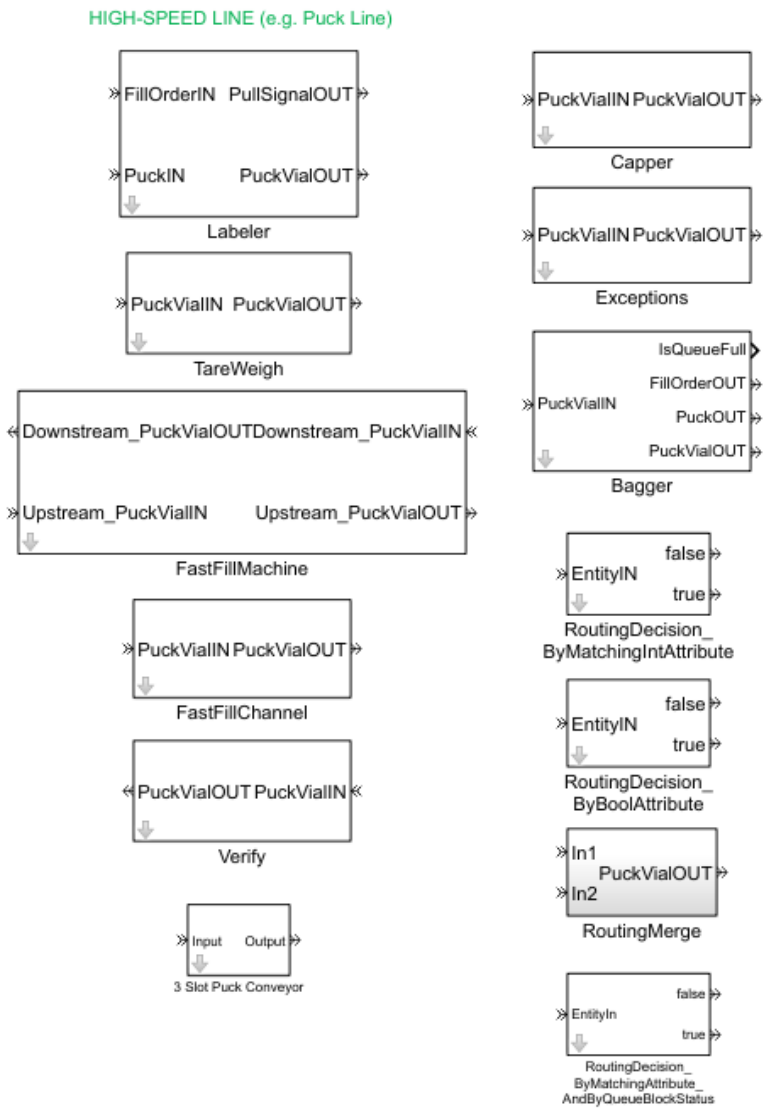

Figure 7-4 High Speed Resource Library

The HighSpeed subsystem models both the flow of pucks through fulfillment resources and control decisions associated with sequencing release of order lines and their association with specific pucks. The puck conveyor system is modeled not as a single unified subsystem, but as individual conveyor components connecting fulfillment resources, that execute transport, divert and merge operations on individual pucks. This is possible because once a puck is assigned an order line, it is given a set of attribute values corresponding to the locations it must visit, including whether it goes to a bagger or to the VTS. At each possible divert, the attributes are read, and if the attribute associated with that divert is set, the puck is diverted.

Figure 7-5 shows the HighSpeed subsystem model. Pucks flow from the labeler to the FillSystem, CappingSystem, ExceptionsSystem, and to a routing switch, where they either continue to the

BaggingSystem or are diverted to the VTS. Operational control in the HighSpeed subsystem consists of assigning order lines to the LabelingSystem to dispense and label a vial, insert it into a puck and tare weigh the puck. The LabelingSysem "pulls" orders from the OpCONTROL_HighSpeedSystem whenever one of the labelers is idle. In the current implementation, order lines are selected in the sequence released by the FulfillmentController although other logic could be implemented.

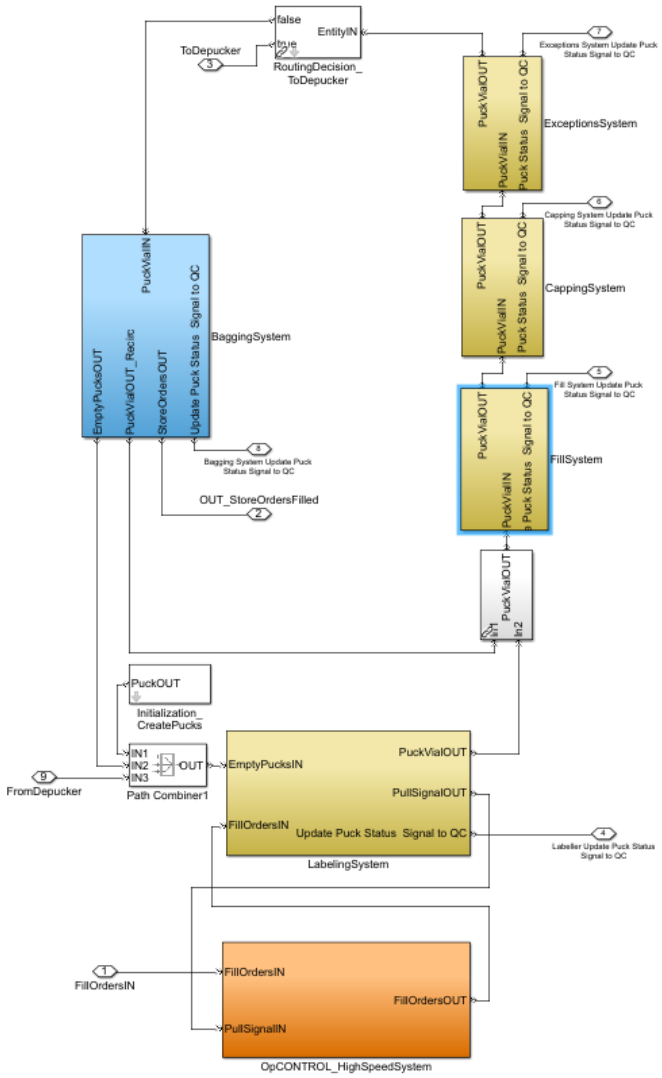

Figure 7-5 HighSpeed Subsystem Model

The model of the BaggingSystem allows for an assigned bagging station to be blocked, i.e., its associated puck queue is full, so pucks would recirculate through the FillSystem, CappingSystem, etc, returning to the bagging station until the queue has available capacity.

The model in Figure 7-5 may seem simple, but that is because the FillSystem block is itself a rather complex subsystem, as shown in Figure 7-6. It consists of three "fingers" or conveyor spurs, each containing three machines, and each machine containing of six individual feeders. Each feeder has a small queue for pucks

waiting for their vials to be filled. The attributes associated with a puck entity allow the entity to be routed to the appropriate finger, and then routed to the appropriate feeder within the finger. If the queue for that

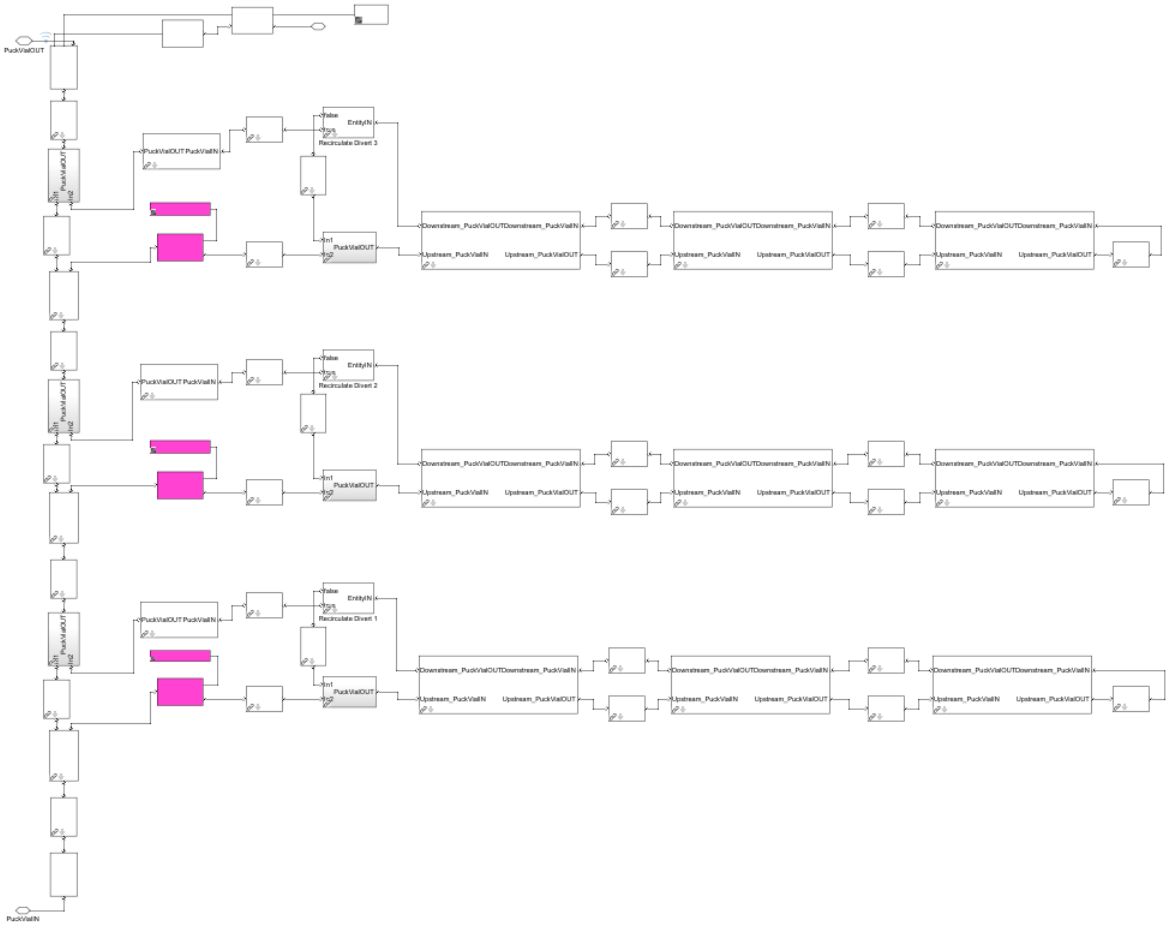

Figure 7-6 HighSpeed FillSystem Model

feeder is at capacity, the puck will recirculate in the finger, if possible, but if the entire finger is blocked, the puck will recirculate all the way through the bagger stations and back to the FillSystem.

7.3.2 HighFlex Model

As with the HighSpeed subsystem, the HighFlex subsystem explicitly represents the fulfillment resources identified in Chapter 6, using library blocks developed specifically for modeling central fill pharmacy resources. The library for HighFlex blocks is shown in Figure 7-7. Note that many of these blocks are subsystem masks, meaning they contain a subsystem model and specify parameters that can be set for the subsystem.

The HighFlex subsystem models both the flow of totes through the fulfillment resources and the control decisions that associate a tote with a specific customer order. Note that there is no sequencing decision for the HighFlex subsystem because customer orders are released one at a time by the Fulfillment Controller. As with the HighSpeed subsystem, the tote conveyor is not modeled as a unified subsystem, but rather as

individual conveyor components connecting fulfillment resources. When an order line is assigned to a tote, the tote entity is given a set of attributes indicating the stations it must visit and what must be dispensed at each station, including the vial transfer system. At each conveyor segment that represents a potential divert, the attributes of the tote entity are read, and if one of them indicates the tote should be diverted then it is diverted.

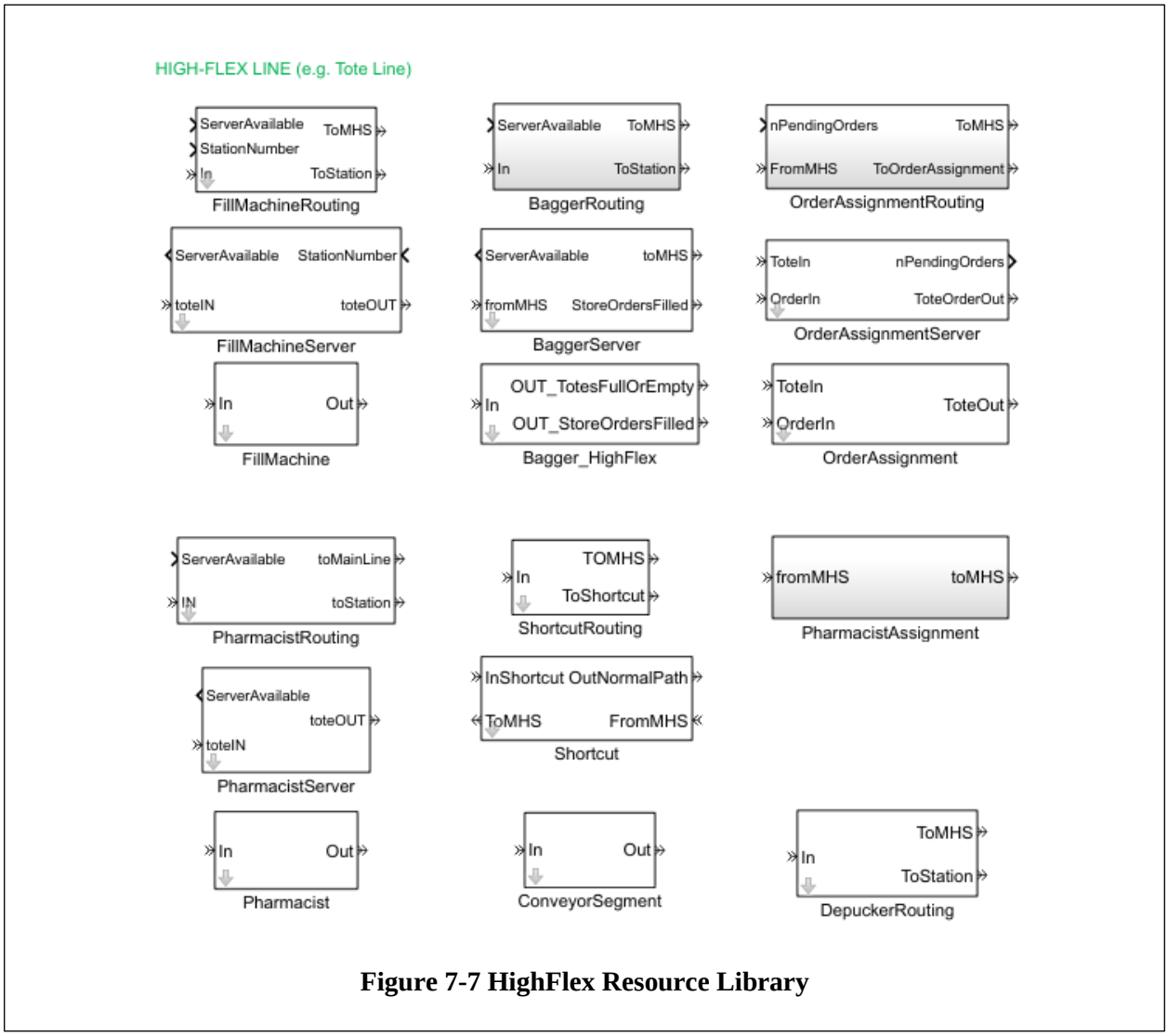

Figure 7-7 HighFlex Resource Library

The HighFlex subsystem model does not have the same kind of hierarchical structure as the HighSpeed subsystem. In the DemoCFP there are 34 different fulfillment stations, counting the robotic stations, the manual fill stations, the pharmacist stations and the bagging stations. There also is the interface to the vial transfer system. Figure 7-8 shows the two "ends" of the HighFlex subsystem, simply to illustrate the basic flow structure. Empty totes are married to customer order at the right side of the figure and basically flow in a clockwise loop through the fulfillment stations. At the left end of the figure is a spur that goes to the vial transfer station. At every fulfillment station, there is a tote queue, which a tote bypasses if it does not need to visit the station or if the queue is full.

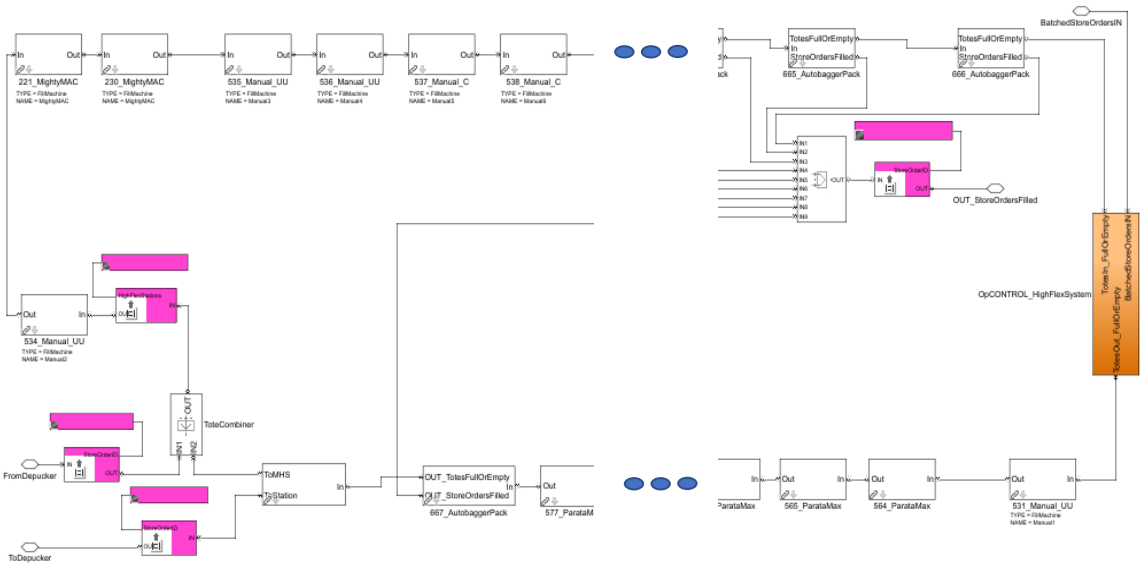

Figure 7-8 HighFlex Subsystem Model (partial)

7.3.3 Vial Transfer System Model

The Merge_HighSpeedToHighFlex model is a simplified implimentation of the VTS model in Chapter 6. There is no explicit vial store, but rather a table that records which order lines are currently stored in the VTS. The maximum size of the table is an input parameter. The robot operations are modeled as a simple delay. There is a finite queue for tote entities but an infinite queue for puck entities. The puck and tote entities are served in FCFS sequence.

7.4 Initial Experimentation

This project has been fortunate to have access to almost 9 weeks of operational data on customer orders and the timing of receipt and other key operations including labeling and bagging. As always, there have been challenges with using "real" data. Simple issues, such as multiple NDCs for the same drug, can present major challenges to a research team with limited access to the data sources.

The arrival rate of orders has a significant impact on operational performance and should be a significant consideration in designing the control system. An analysis of the nine weeks of data regarding the distribution of arrival times is summarized in Figure 7-9. A relatively small fraction of orders are received overnight, but there is a significant "surge" of orders shortly after daily operations begin. The consequence is that the observed workload during the period from 9 am to 11 am is perhaps 2.5 times the observed workload for the remainder of the day. What this suggests is the need for an operations management strategy that can smooth the workload over the day without jeopardizing the service level, i.e., the fraction of eligible orders that are completed in time to be shipped to the store on the same day.

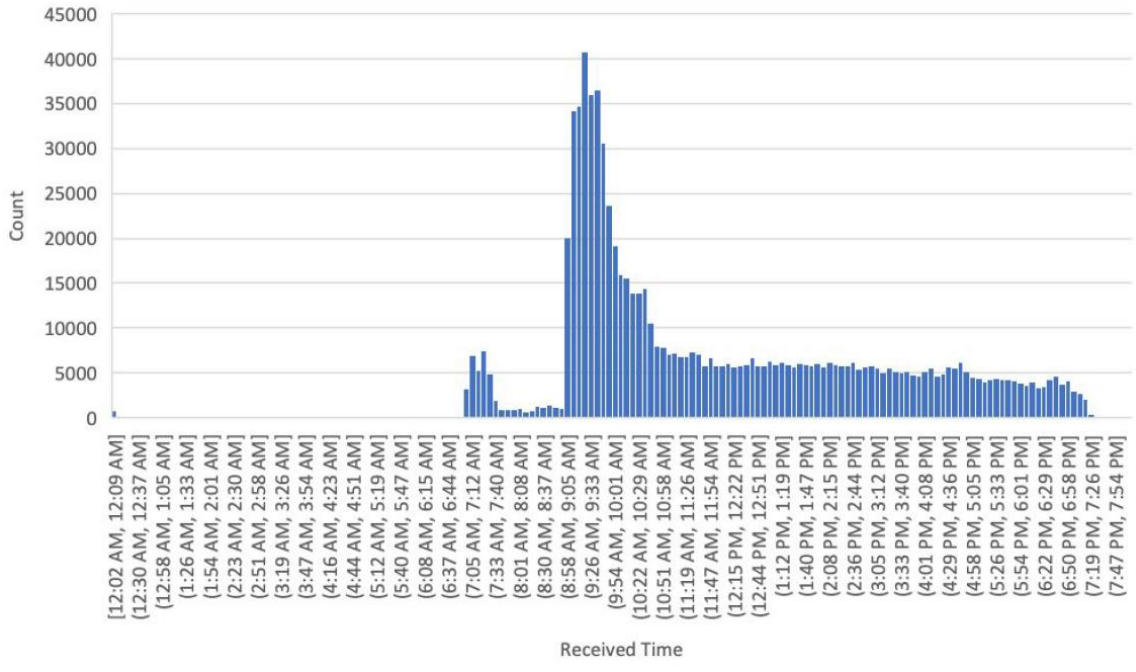

Figure 7-9 Distribution of Order Received Time

Not surprisingly, because the existing system basically releases orders as fast as possible, the average cycle time is relatively larger early in the day and diminishes over the day as order volume subsides. This is

illustrated in Figure 7-10. Not only does the arrival rate of order diminish over the day, the large work in process due to the surge of orders from the early part of the shift gets worked down. Again, this suggest a need for an operations management approach that can balance the workload over the day.

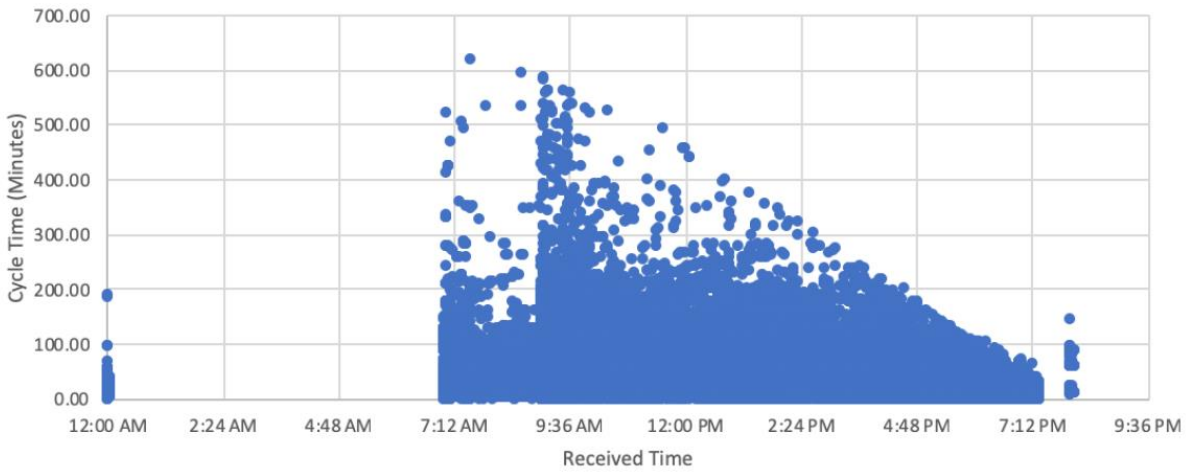

Figure 7-10 Cycle Time vs Received Time

The desire to exploit this large realistic dataset prompted the development of a Python-based GUI to simplify the specification of parameters, selection of datasets and assembly of computational results. One study looked at all 50+days (the CFP did not operate on Saturdays), ran the simulation for the arriving orders and compiled cycle time data as illustrated in Figure 7-11. A few of the daily results are eliminated from the chart because of simulation issues, specifically that a significant number of orders did not exit the system. An interesting observation is the four outlier points where cycle time is two to three times larger than other cycle times for the same throughput. Closer examination reveals that these four days all are Sundays, and the increased cycle time is due to orders on the HighFlex system, where there are manual workstations. The most likely explanation is that on Sundays, the staffing of these manual workstations is much lower than it is on other days, so their utilizations are much higher, causing much more work-in-process, which from Little's Law would indicate much longer cycle times for a given throughput level. For the other days, where standard staffing likely applies, cycle time increases as throughput increases, as one would expect.

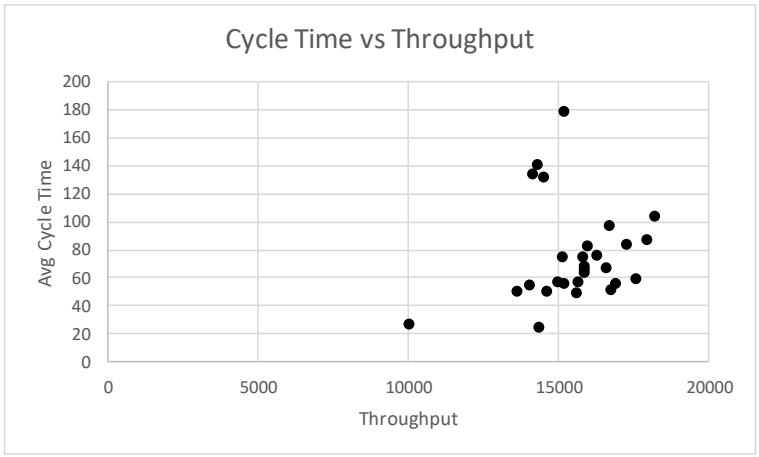

Figure 7-11 Simulated Cycle Time vs Throughput

The simulation model allows the user to specify the numbers of pucks and totes in the system. These are expected to have at least some influence on cycle time and perhaps on throughput as well. One set of experiments looked at the impact of varying the numbers of pucks and totes. Figure 7-12 illustrates the impact of varying the number of pucks in the HighSpeed system and represents a simulation over a fixed period of time with the number of totes in the HighFlex system fixed at 200. If there are not enough pucks, the work-in-process is limited and for a given capacity, the throughput also is limited. As the figure shows, throughput (during the fixed time interval) will increase as more pucks are added, up to a saturation level, in this case, around 400 pucks. Interestingly, the figure also indicates that throughput might actually decline if there are too many pucks in the system, as they all must be somewhere, and create congestion.

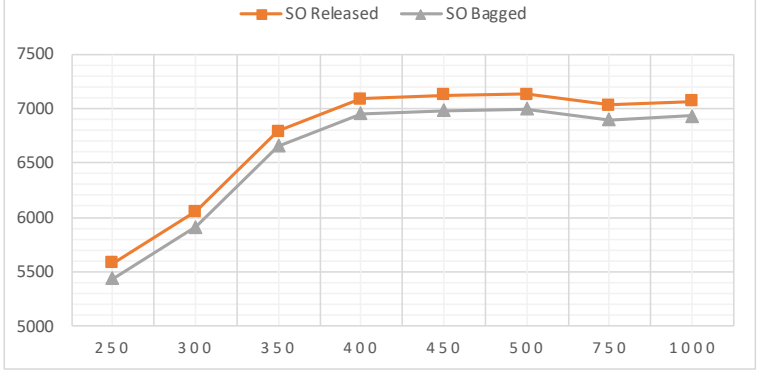

Figure 7-12 Throughput vs Number of Pucks; 200 Totes

The impact of increasing the number of totes is shown in Figure 7-13. When there are 250 totes, increasing the number of pucks beyond the saturation level of 400 actually causes throughput to become erratic. While it is not completely clear why this would happen, it likely is a consequence of interaction between the numbers of carriers in the system, the size of the order release batches, and the mechanism for releasing each individual order into the fulfillment processes.

Figure 7-13 Throughput vs Number of Pucks; 250 Totes

Clearly, there is a great deal that might be learned from more extensive experimentation with the DemoCFP simulation.

7.5 Modeling Issues

There are a number of challenges in transitioning from the relatively platform independent system model, rendered in a standardized system modeling language to the relatively platform specific simulation model rendered in a particular DES language.

In the system model, there are both object flows and control flows between activities. In Simevents, object flows are represented as entity flows, and there can be a very close correspondence between object flows modeled as blocks and entity flows modeled as entities. Control flows from the system model also become entity flows in the Simevents model, and typically require "splitting" an entity representing either an object or another control flow into two entities, one still representing the original entity flow and the other representing the desired control flow.

A related issue is the representation of controller behavior. In the system model, a controller is an object (block) with its own identity and sets of behaviors—modeled as activities—representing decision-making or state model management. These behaviors can be invoked by calling an operation of the controller for which the activity is a method and these invocations can appear in multiple process models with no ambiguity or confusion; every invocation refers to the same controller block and the associated decision logic or plant model manipulation can be arbitrarily intricate. In Simevents, simple control decisions, such as queue dispatching, are associated with a specific queue. If you want to change the controller logic, you must locate every corresponding queue and modify it. Controller decisions that are more than local queue dispatching and plant model maintenance processes can be modeled only by escaping to MATLAB. In the current implementation of the DemoCFP simulation, each controller behavior is a stand-alone MATLAB function, i.e., there is not a "controller object" in Simevents that corresponds to the controller block in the system model. As a consequence, the controller model is fragmented as MATLAB escapes and may appear in many locations. Unless a very disciplined block naming convention is followed, it can be challenging to determine exactly what controller is being invoked by one of these escapes.

Similarly, modeling the transforming behavior of active resources is straightforward if the transformation is merely a delay, and can be modeled with one of the standard "queue" blocks. However, if the transformation requires changing the plant model, e.g., availability of resources or changing the identity of a part, then several steps are involved. The entity representing the input to the process is split, creating a control flow that is used to invoke a MATLAB escape where the plant model can be updated, or a new entity created with new attributes.

Conveyor modeling presents some particular challenges, not just for Simevents but for almost all COTS simulation languages. The behavior of a conveyor in the DemoCFP system model is simply an activity, move(origin, destination) and that captures the general capability of the conveyor. In the simulation model, representing that behavior must represent both the location state change and the time required for the state change. In the current version of the DemoCFP simulation model, conveyor segments are represented by an "N server queue" where N represents the number of flow entities that can be in motion at one time and the server processing time is the time to traverse the conveyor segment. It should be noted that the original development of the DemoCFP was started in a version of Simevents that now is two major releases out of date. In the most recent version of Simevents there is a standard library object modeling conveyor segments with fixed speed, fixed carrier size and fixed interval between carriers. Any revision or further development of the DemoCFP as a testbed should explore this newer modeling capability.

A further issue related to the modeling of conveyors is that in the system model, carriers always remain on the conveyor, and when a conveyor move operation is completed the appropriate controller is notified and invokes the desired behavior at the workstation to which the carrier has been delivered. This architecture allows for operations management decisions that can consider the states of system parts other than the conveyor and the workstation to which the carrier has been delivered. In contrast, the implementation in the DemoCFP simulation has the entity corresponding to the carrier (either puck or tote) exchanged between blocks representing the conveyor and the workstation, and the only information the workstation has is contained in the attributes of the carrier entity. This significantly limits the scope of any decision making at the workstation.

7.6 Future Simulation Model Development

The DemoCFP simulation is a large and complicated simulation. It has been developed over a period of three years, with significant contributions from at least five developers. While it goes a long way to ward the goal of creating a testbed for experimenting with smart operations management, there are a number of opportunities for improvement.

An obvious improvement would be to upgrade the model to the latest release of the modeling platform, which would provide some improved modeling capabilities and also probably significantly improve the runtime performance.

Improvements to the Python-based user interface would significantly reduce the time and effort required to conduct parametric studies of key design parameters, such as carrier counts, batch sizes, VTS capacity, etc.

Finally, there is need for a simulation modeling approach that mirrors the active resource modeling approach in the DemoCFP system model. There an active resource has both resource and controller part properties. There are clearly identified interfaces for the active resource in terms of both object flows and control flows. This enables a nicely object-oriented approach to system specification. This is missing in the current implementation of DemoCFP where there is not a unified representation of active resources that integrates their resource and control parts. With such a unified representation, it also should be possible to have the Matlab escapes for control decision making more closely conform to the control decision framework outlined in (Sprock, Bock, & McGinnis, 2019).

8 Conclusions

The fundamental thesis of the research reported here is that achieving the full realization of smart manufacturing operations management will remain elusive until two fundamental requirements are met:

1. The manufacturing system can be specified with the same degree of completeness and precision as can the products that it produces, and
2. There is a generic and broadly reusable architecture or pattern for the structure and behavior of manufacturing operational controllers.

Meeting the first requirement requires a manufacturing system reference model providing both an ontology and a syntax enabling the creation of computational representations of manufacturing systems. Meeting the second requires developing and demonstrating smart manufacturing operations controllers.

The work reported here does not completely meet these two requirements, but it makes a significant contribution toward them. The approach and some of the tools employed have been motivated by, informed by, and significantly improved by recent developments in Model-Based Systems Engineering as applied to the design and development of space missions ((Bayer) and aircraft programs ((Sheeleey, 2014). The foundation for the work includes the ISA-95 standard for planning and control ((The International Society of Automation, 2019) and two recent dissertations ((Thiers, 2014), (Sprock T. A., 2016)).

Chapter 2 briefly introduces the concept of discrete event logistics systems, the basic elements of the reference model, and the conceptual framework for a generic operational controller. Much more detail may be found in (Sprock, Thiers, McGinnis, & Bock, 2019). The chapter also demonstrates the relationship between the DELS reference model and the basic elements of the modeling framework supporting the ISA95 specification, in particular, that the DELS reference model generalizes some of the key elements of the ISA95 reference model. This is important, because it establishes the DELS reference model not as replacing ISA95 but as extending it.

Chapter 3 is a deeper dive into the implications for operational controller requirements, functions and architecture. Key elements include the notion of event-driven control decisions, fundamental behavior of active resources, the fundamental concepts behind transfer of product between active resources, and the importance of a plant model in the controller architecture.

The demonstration use case is presented in Chapter 4. It is a large-scale, highly automated central fill pharmacy. The presentation is based on publicly available information, and is the basis for the particular CFP, referred to as DemoCFP that is subsequently modeled and simulated.

Chapters 5 and 6 use the OMG SysML™ as implemented in MagicDraw™ to create a computational representation of the DemoCFP system that explicitly represents the product flowing through the system, the active resources that transform the product, including material transport, and the operational controllers responsible for batching orders, releasing orders and managing transportation. It appears that this is the only such production system model currently available in the public domain. There are a number of journal and conference papers that mention SysML and manufacturing, but the focus is almost always on a specific manufacturing process, rather than operational control of the entire system, or addresses the issue at a very high level of abstraction. Among the contributions of these two chapters are: (1) a reusable pattern for organizing the DELS model; (2) the explicit and formal integration of activity

models of processes, and invocable behaviors of resources and controllers; (3) the controller as a part property of the active resource (domain) whose part property active resources are managed by the controller.

Chapter 7 presents a discrete-event simulation model for DemoCFP, rendered in the SimEvents™/MATLAB™ platform, and based on the SysML model from Chapter 6. The simulation model allows experimentation with some parametric aspects of the DemoCFP specification, such as batch sizes, numbers of material transport carriers, and capacity of some active resources. The chapter contains some observations about the challenges of going from a computational but analysis-agnostic system model to a simulation analysis specific model.

These chapters break new ground in the ongoing development of smart manufacturing and provide important contributions to addressing the two fundamental requirements—manufacturing system modeling and generic operational controller architecture. Much remains to be done. The current state of DELS system modeling does not yet have a "best practice" for handling the transition from defining a class of systems to defining particular systems. More DELS use cases are needed addressing other types of DELS, including fabrication, assembly and services. The shortcomings of available COTS discrete event simulation packages in modeling control need to be more fully catalogued and defined, leading to improved simulation tools and perhaps to automation in creating simulation models from system models.

9 References

Bayer, T. J. (n.d.). *Model Based Systems Engineering on the Europa Mission Concept Study.* Retrieved from https://trs.jpl.nasa.gov/bitstream/handle/2014/45000/11-5594_A1b.pdf?sequence=1

Cho, H., Son, Y., & Jones, A. (2006). Design and conceptual development of shop-floor controllers through the manipulation of process plans. *International Journal of Computer Integrated Manufacturing*, 359-376.

Davis, W., Jones, A., & Saleh, a. A. (1992). Generic architecture for intelligent control systems. *Computer Integrated Manufacturing Systems*, 105–113.

Dilts, D., Boyd, N., & Whorms, H. (1991). The evolution of control architectures for automated manufacturing systems. *Journal of manufacturing systems*, 79–93.

Galloway, B., & Hancke, G. (2013). Introduction to industrial control networks. *Communications Surveys & Tutorials, IEEE*, 860–880.

Jans, R., & Degraeve, Z. (2008). Modeling industrial lot sizing problems: a review. *International Journal of Production Research*, 1619–1643.

Sheeleey, B. ,. (2014). *MBSE Implementation Across Diverse Domains at The Boeing Company.* Retrieved from https://www.omgwiki.org/MBSE/lib/exe/fetch.php?media=mbse:02-iw14-mbse_workshop-mbse_implementation_across_diverse_domains_at_boeing-carsonmalonepalmersheeley.pptx

Silver, E., & Peterson, R. (1998). *Inventory management and production planning and scheduling.* John Wiley and Sons.

Smith, J., Joshi, S., & Qiu, a. R. (2003). Message-based Part State Graphs (MPSG): a formal model for. *International journal of production research*, 1739–1764.

Sprock, T. A. (2016). *A Metamodel of Operational Control for Discrete Event Logistics Systems.* Retrieved from https://smartech.gatech.edu/handle/1853/54946

Sprock, T., Bock, C., & McGinnis, a. L. (2019). Survey and classification of operational control problems in discrete event logistics systems (DELS). *International Journal of Production Research*, 5215-5238.

Sprock, T., Thiers, G., McGinnis, L., & Bock, C. (2019). *Theory of Discrete Event Logistics Systems (DELS) Specification.* NIST Interagency/Internal Report. Retrieved from https://doi.org/10.6028/NIST.IR.8262

The International Society of Automation. (2019, November 21). *ISA95, Enterprise-Control System Integration.* Retrieved from https://www.isa.org/isa95/

Thiers, G. G. (2014). *A Model-Based Systems Engineering Methodology to Make Engineering Analysis of Discrete-Event Logistics Systems More Cost-Accessible.* Retrieved from https://smartech.gatech.edu/bitstream/handle/1853/52259/THIERS-DISSERTATION-2014.pdf

Vogel-Heuser, B., Witsch, D., & Katzke, U. (2005). Automatic code generation from a UML model to IEC 61131-3 and system configuration tools. *ICCA'05. International Conference on Control and Automation*, 1034–1039.