Proceedings of the 2017 Winter Simulation Conference
W. K. V. Chan, A. D'Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer, and E. Page, eds.

INCORPORATING ABSTRACTION METHODS INTO SYSTEM-ANALYSIS INTEGRATION METHODOLOGY FOR DISCRETE EVENT LOGISTICS SYSTEMS

Timothy Sprock
Conrad Bock

National Institute of Standards and Technology
100 Bureau Dr, MS8260
Gaithersburg, MD 20899, USA

ABSTRACT

Analysis models, such as discrete event simulation models, are used to support design and operation of discrete event logistics systems (DELS). The time and expertise required to construct these analysis models can be significantly reduced by automatically generating them from formal models of the systems being analyzed. DELS analysis models can be constructed from system abstractions much more reliably when the system and analysis are specified at compatible levels of abstraction. Formal modeling languages, such as those used in object-orientation, make abstraction explicit, simplifying the mappings between system and analysis models and increasing reusability of the integration. In this paper, we propose fundamental abstractions for DELS and identify corresponding libraries of analysis models. These are used in a system-analysis integration methodology that incorporates abstraction as an explicit step, providing a path to refine and extend those abstractions and model libraries to generate analysis models.

1 INTRODUCTION

Discrete event logistics systems (DELS) are dynamic systems that transform objects flowing through networks of interconnected resources (Mönch et al. 2011). These include systems such as supply chains, manufacturing systems, transportation networks, warehouses, and health care delivery systems. Traditionally, each specialized kind of DELS has been addressed with its own dedicated research and development. However, these systems share a common abstraction — *products* flowing through *processes* being executed by *resources* configured in a *facility* (PPRF) — and they appear together in integrated models of engineering enterprises. For example, production systems might integrate storage and fulfillment capabilities as well as material handling and transportation systems, and supply chains might integrate flows between warehouses, transportation systems, and manufacturing or health care facilities.

Industrial and systems engineers use a broad variety of analysis methodologies to support design and decision making in DELS. Many, if not all, of these methodologies require significant time and expertise to manually construct analysis models, such as discrete event simulation. The methodologies would be simpler and less time-consuming with automatic formulation and construction of analysis models from an independent representation of the system. Multiple analysis models could be generated from a single system model rather than constructing each analysis model by hand, as is commonly done.

Analysis models that conform to standard definitions of the system (and abstractions thereof) can be automatically generated and reused through formal system-analysis integration (SAI) methods. Most analysis models commonly developed for DELS are based on some abstraction (simplification, view) of the system. Therefore, formalized abstractions should be a cornerstone of system modeling and analysis methodologies. Reusable abstractions can be captured in system reference models and analysis libraries.

This paper seeks to harvest these abstractions, make them formal, and automate formulation of analysis models from the abstractions.

This paper presents two modeling contributions that use DELS abstractions to simplify integration of system and analysis models and make the integrations reusable in more applications. The first contribution is a modeling methodology that explicitly incorporates abstraction into a model-to-model (M2M) transformation process (Section 2). Generalization is used to formalize the relationship between system models and abstract models. This method improves the reliability of retrieving abstractions from system models. The second contribution of this paper is fundamental abstractions used for SAI in DELS (Section 3). The result is models at multiple levels of detail that support simple and accurate retrieval of abstractions from specialized system models. Examples of conforming analysis models, such as those that could support multi-fidelity simulation methods, are identified for each level of abstraction.

2 SYSTEM-ANALYSIS INTEGRATION MODELING METHODOLOGY

Manually integrating information about DELS with discipline-specific analyses, such as discrete event simulation, is time-consuming and difficult. Analysis models use concepts and formats that are inconsistent with each other and not integrated with formal system models. Model-based systems engineering methodologies provide model-to-model transformation methods to automate the translation between system models specified in standard, formal languages (Estefan 2007). System-analysis integration extends these methods by developing standard models for widely-used engineering analysis techniques and integrating those models with widely-used systems engineering modeling languages. Then standard transformations can be developed to automatically translate between these standard system / analysis models and widely-used analysis tools.

However, significant barriers exist to reusing knowledge and tools for modeling and analysis integration. There is often not enough commonality among systems or analysis tools, requiring separate transformations for each pair of system and analysis models. Knowledge relating system and analysis models is encoded in informal and manual mappings between them. This implicit mapping knowledge is also a challenge for manual analysis modeling methods. For example, how should engineers decide which analysis component to select from a simulation model library to represent a particular system object?

Relying on complex, ad-hoc mappings to automate generation of analysis models only moves complexity from the models to the mappings. One source of complexity in these mappings is that system and analysis models are often specified at different levels of abstraction. Integrating abstraction as an explicit step in the M2M methodology simplifies mappings between system and analysis models by breaking them up across multiple levels of abstraction. The analysis methodology proposed in Thiers (2014) separates the M2M transformation into two steps: one from the system model to an abstract model, and then a second to the target analysis model. The system model is mapped to the abstract model via stereotype application in the Unified Modeling Language (UML) (OMG 2015). Network models are proposed as reusable abstractions for DELS. Stereotype application, however, cannot guarantee the correctness of resulting abstraction, because stereotypes are not available for specialization to domain-specific models once they are applied, and are only supported in UML tools.

The approach proposed here uses generalization relationships to formalize the results of abstraction, rather than stereotype application. Generalization relates classes of things to broader classes that include them. For example, the class of forklifts is generalized by the class of mobile resources. Generalization enables system models to be constructed (specialized) directly from abstractions, rather than mapped to the abstraction after the system model has been constructed, as with stereotypes. The resulting system model naturally conforms to the abstraction, because the abstraction is identified as the broader class. Abstractions can be retrieved correctly and efficiently from detailed system models. Model libraries and taxonomies constructed using generalization can be extended and specialized to incorporate new specific system behaviors and any corresponding analysis models, while retaining access to higher levels of abstraction.

Generalization is supported in almost all modern programming languages, as well as UML, providing many more potential modeling platforms than stereotypes.

2.1 Modeling Method: Formalizing Mappings to Abstractions with Generalization

Frantz (1995) describes a taxonomy of abstraction methods for simulation modeling. Two of these abstraction methods, model boundary modification and model behavior modification, can be formalized using object-oriented languages. The model boundary modification method creates taxonomies of model elements related by explicit simplifications and assumptions. Object-oriented languages formalize this with generalization. The model behavior modification method aggregates components and behavior into systems and other behaviors, respectively (Frantz 1995). Object-oriented languages formalize this with composition. This paper and this section focus on model boundary modification.

Generalization formalizes the mapping between system models and their abstractions. It is shown in UML by a hollow headed arrow going from a specialized class to a more abstract one. For example, the generalization from *Warehouse* to *Node* in Figure 1 specifies that everything fitting the description of *Warehouse* will also fit the description of *Node* (a network abstraction). This formal relationship between two model elements guides construction of the transformation from the system model (of warehouses) to the abstraction (of network nodes).

Methods that transform specialized classes into abstract ones, such as conversion, up-casting, or other substitution methods, require an explicit definition of transformations from detailed properties in system models to less detailed properties in abstractions. These transformations are specified by expressions captured as mathematical equations or UML constraints. For example, consider the problem of selecting locations for new warehouses within a supply chain. Algorithms solving facility location problems (FLP) use a network formulation (abstraction) to select the best set of nodes to optimize the network configuration. Formulating this analysis requires mapping *Warehouse* to the more general class *Node*. It also requires extracting the cost of selecting each node from a more detailed cost model included in *Warehouse* (Figure 1). For example, the *fulfillmentCost* and *orderManagementCost* of warehouses are aggregated to *totalCostToServe*, which then generalizes to the *cost* of a Node.

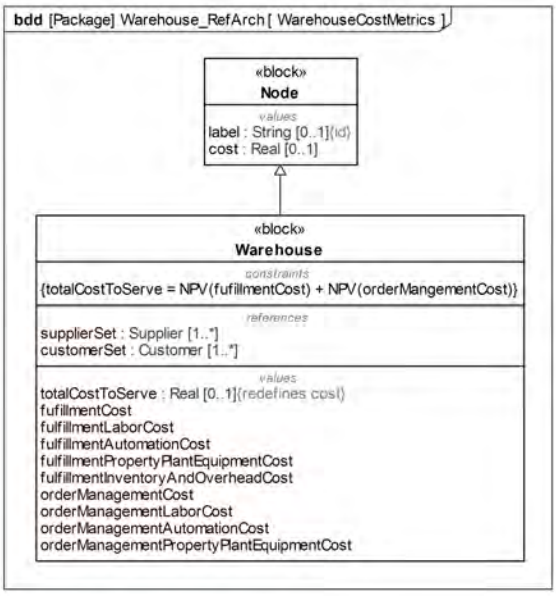

Figure 1: Warehouses are specialized nodes. Detailed cost properties of warehouses are mapped to the cost property of nodes.

Generalization is a method to organize things into taxonomies (classifications) by their similarity, defining specialized classes to elaborate differences within broader classes while retaining a relationship to them. Taxonomies constructed using generalization explicitly model the assumptions, extensions, and simplifications embedded in them. Things that are logically similar can be organized by generalization. For example, trucks and forklifts can be generalized to mobile resources that carry pallets, mobile resources in general, or all resources. Classes can be specialized to capture differences between specialized things. For example, machines that execute subtractive manufacturing processes can be specialized into classes of milling machines and turning machines, or further into specific brands of milling or turning machines. A model library of extensible components simplifies the construction of analysis models by reusing similar mappings and transformations. In DELS, similarities between system models reduce the number of model library components that need to be constructed in the analysis language. These similarities can also be exploited to reuse analysis models that apply to the abstraction, rather than the system model, when that level of detail is not necessary or available.

One of the key advantages of this SAI method is that system models are constructed from abstractions, rather than mapped to abstractions after modeling. The mapping is defined as part of system model construction by extending existing reference models using generalization. This approach reduces errors and inconsistencies resulting from ad-hoc abstraction mappings and reduces the time-required to verify correctness of abstractions retrieved from the system model. The abstraction used to construct the analysis model can be extracted correctly and automatically from the detailed system model.

3 ABSTRACT MODELS FOR DELS

Analysis models for DELS are formulated at multiple levels of abstraction, due to variations in the information available about the system and in desired performance of the solution method. A modeling methodology should support construction of abstract analysis models, but also provide a method to increase fidelity of those models. Methods for abstraction, refinement, and reuse of models and model components are not supported by most analysis languages, such as those used to construct simulation and optimization models.

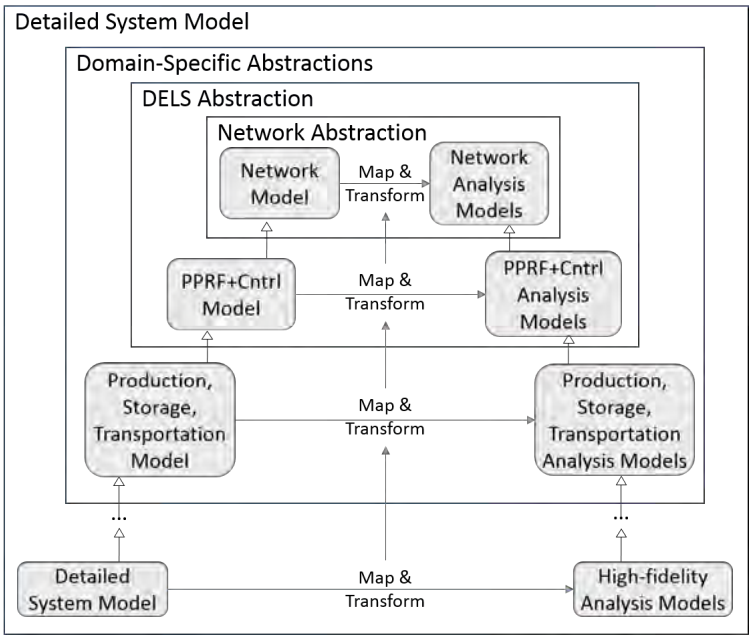

Figure 2: A hierarchy of system and analysis models represent the real-world system at multiple levels of abstraction.

Reusability of the SAI method in the previous section relies heavily on constructing the right abstractions. If they do not apply to a broad enough class of systems, then the return on investment for abstract modeling and integration is not substantially greater than integrating each system individually. A more robust solution is to use several inter-related abstractions of DELS, rather than a single abstraction. The proposed DELS abstractions are multi-layered and designed to maximize reuse of analysis models by harvesting commonalities across this class of systems and their related analyses. Object-orientation provides a method for extending abstractions to specify domain-specific attributes and specialized analysis models and methods. Rather than having a two-step transformation with a single abstract model, a continuum of models can represent the real-world system at many levels of abstraction (Figure 2). This continuum of system abstractions and associated analysis models reduces or eliminates the need for ad-hoc models or transformations to bridge the gap between the two. This section describes reusable, standard, multi-layered abstractions for DELS and some typical analysis models applicable at each level.

3.1 Network-based Abstractions

Network-based abstractions are common in DELS modeling because of their widely-understood mathematical representations, their suitability to many algorithms, and their applicability to a broad range of (abstract) analysis questions about the structure of and flows through DELS. Basic networks, flow networks, and process (or queuing) network models are foundational abstractions for DELS (Figure 3) (Thiers 2014). Each of these abstractions has libraries of analysis algorithms, such as finding shortest paths and optimal facility locations (Drezner and Hamacher 1995), determining throughput for multi-commodity flow networks (Ahuja et al. 1993), as well as service time and utilization in queueing networks (Walrand 1988, Marzolla 2010).

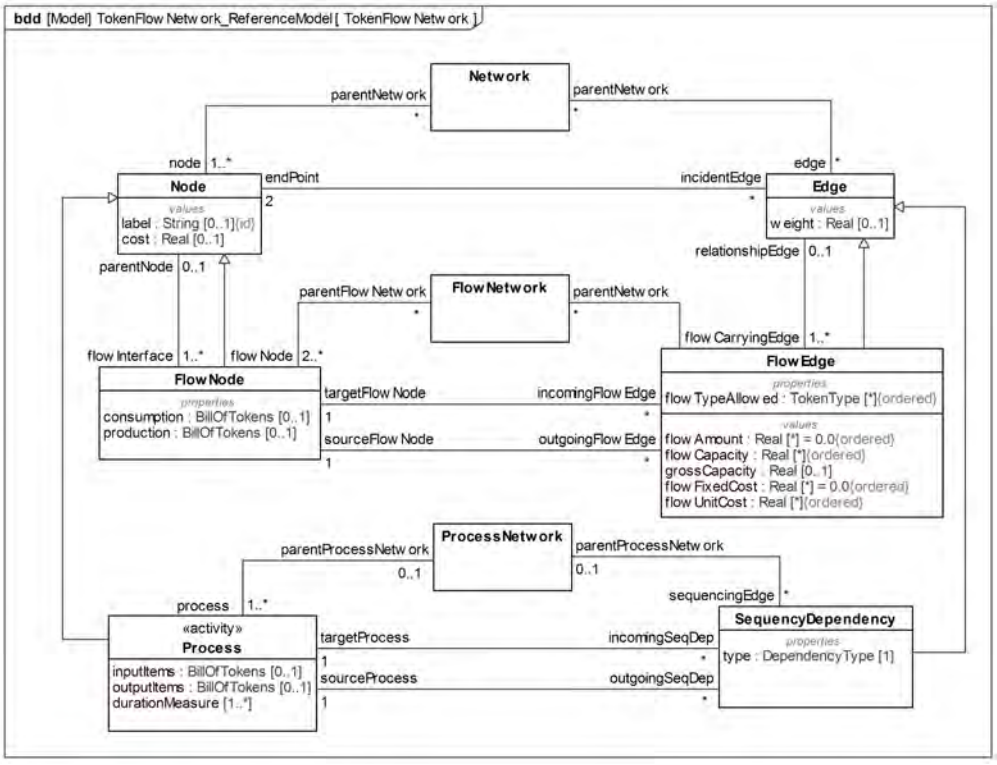

Figure 3: Basic networks, flow networks, and process (or queuing) networks are foundational abstractions for DELS.

3.2 DELS Behavior

Integrating system and analysis models is simpler when the systems models extend standard abstractions corresponding to widely used analysis techniques. DELS share a common abstraction for their behavior: products flowing through processes being executed by resources configured in a facility (PPRF) (Figure 4). Each element of the PPRF abstraction is elaborated with taxonomies of more detailed abstractions, such as the resource taxonomy in OZONE (Smith and Becker 1997) or the resource/process taxonomy in ISA-95 (ISA 2010) (Figure 5).

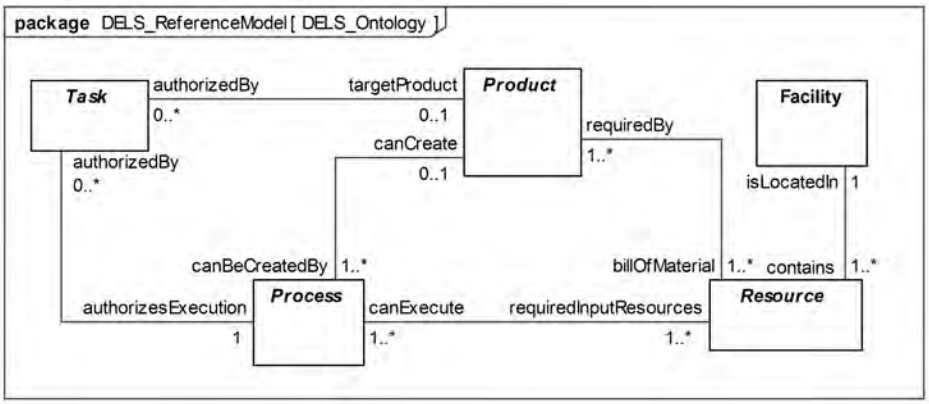

Figure 4: DELS share a common abstraction for their behavior: products flowing through processes being executed by resources configured in a facility.

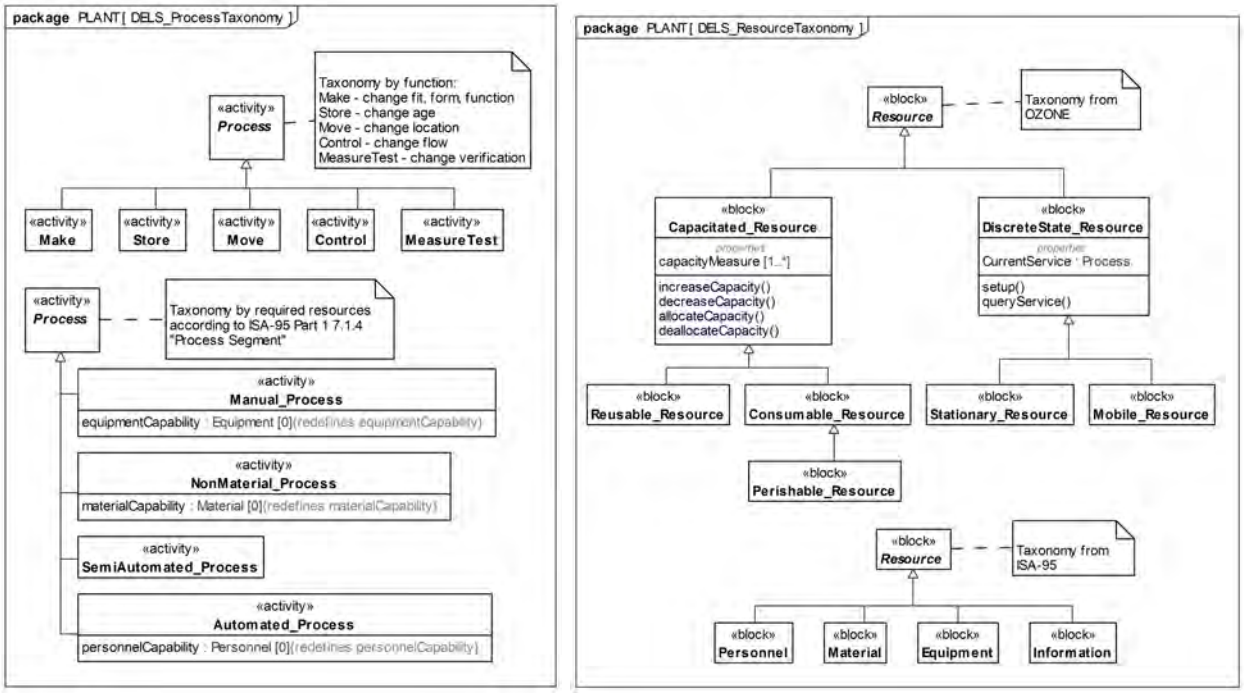

Figure 5: Each element of the PPRF abstraction is elaborated with taxonomies of more detailed abstractions.

Analyzing the complex interactions inherent in DELS behavior is usually done using simulation, such as discrete event. For this class of analysis models and tools, system models constructed from a standard

model library of components specialized from PPRF make it easier to validate simulation models and replicate the results. The application of analysis methods designed for one DELS subdomain to another can be formalized by mapping the behavior of systems from the two subdomains to a common abstraction beforehand. For example, analysis methods for solving the traveling salesman problem (TSP) can be used to construct manufacturing schedules (Lenstra and Kan 1975), and methods for scheduling manufacturing job shops can be applied to scheduling nurses or operating rooms in a hospital (Pham and Klinkert 2008), (see also (Beck et al. 2003) for a comparison of vehicle routing (VRP) and job shop problems (JSP)).

Taxonomies of analysis models for specialized system behaviors should be differentiated from taxonomies of solution methods for making control decisions. For example, extensions to common analysis models, such as flexible JSP (FJSP) or capacitated VRP (CVRP), are constructed from specializations of (extensions to) abstract system behavior models (the PPRF taxonomies). Adding resource flexibility to a process, or capacity constraints to resource (system behavior), does not change the types of control decisions that need to be made (control functions), just how decision is made (analysis methods).

3.3 DELS Operational Control

Operational control manipulates the flow of tasks and resources through a system in real-time, or near real-time. A significant body of analysis models and algorithms exist to select control actions to be taken, including: 1) "Should a task be served?" (*admission*); 2) If so, "When should the task be serviced?" (*sequencing*); and, 3) "By which resource?" (*assignment*); 4) "Which process does the task require next?" (*dynamic process planning/routing*); and 5) "To which state does a resource need to be changed?" (change in *resource capacity* or *capability*). Our model of operational control links a controller, which may use analysis models to support decision making, to actuators in the base system (plant) that execute the prescribed actions (Figure 6) (Sprock 2015).

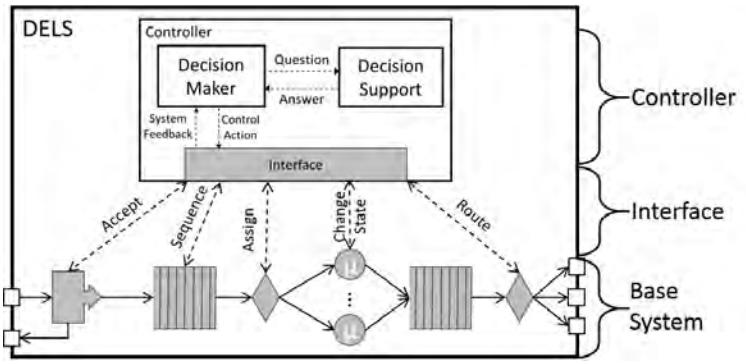

Figure 6: A model of operational control that links a controller to actuators in the base system that execute the prescribed actions.

Each control function is carried out in the base system by a specialized resource (actuator that executes the selected action) performing a specialized process (how the actuator executes the selected action). Process and resource taxonomies are extended to model control functions in the plant (Figure 7). System specifications use these model library elements to indicate *where* and *when* control decisions are required by the system. Decision support is designed separately, specifying *how* control actions are chosen.

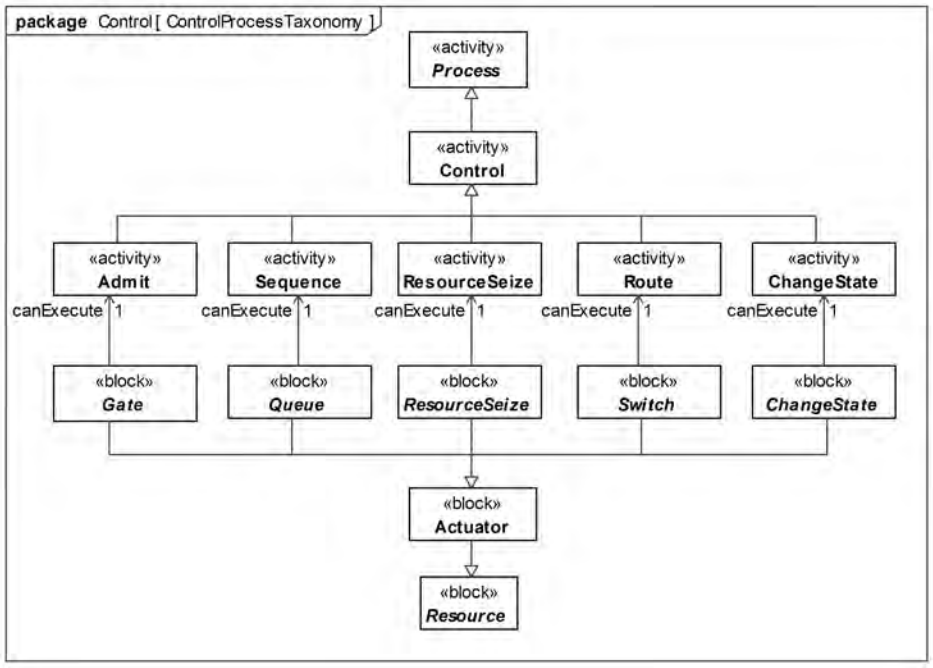

Figure 7: Each control function is carried out in the base system by a specialized resource and specialized process.

A wide variety of specialized system resources can carry out each control function, for example, diverters, robotic arms, or humans can admit tasks and raw materials into the system. However, most analysis models do not require highly detailed specification of every resource in the system. Detailed specifications of actuators that execute control functions are mapped by generalization to common abstractions (specialized resources) that specify the control function for each class of actuators. These abstractions of actuators are used to construct analysis models.

Decision support methods for each control function can also be specialized (refined) from a library of simple methods or can be abstracted (simplified) for analysis models. For example, sequencing tasks in a queue may be specified by a simple static priority rule that can be later refined into a complex decision tree of priority rules. Every control function in the system conforms to the same definition and uses the same abstraction. A standard abstraction of DELS control functions enables consistent and correct models of control across analysis models and tools, such as discrete event simulation.

3.4 Extending and Composing DELS

DELS abstractions can be specialized to represent many kinds of DELS, reusing the libraries described in previous sections as needed. The *Process* element can be specialized into a taxonomy of basic DELS functions, *Make*, *Store*, and *Move*. These processes are allocated to specialized DELS for production, storage, and transportation, respectively (Figure 8). Allocation of a process to a DELS, such as allocating *MOVE* to *TransportationSystem*, adds an operation (function) that executes that process when the operation is invoked. The DELS must provide a behavior that implements the function by defining what process steps and required resources are used to execute that operation.

Many DELS are composed of other DELS. For example, a manufacturing facility might have production systems, such as work cells or production lines, linked by material handling or transportation systems, and buffered by intermediate storage points. Composing DELS out of existing specialized DELS, rather than defining a new monolith, allows domain-specific analysis models to be reused for functionally similar

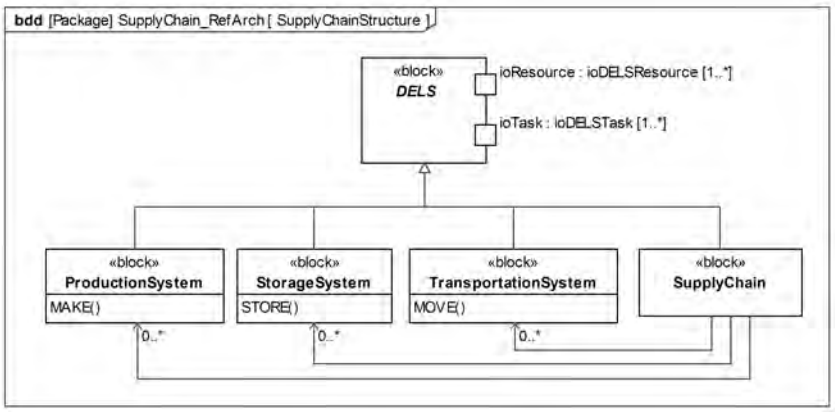

Figure 8: The DELS abstraction can be specialized to represent many kinds of DELS. These specializations can be composed to specify complex DELS.

subsystems. Each specialized kind of DELS can be further specialized as necessary to capture domain-specific features, such as differences between manual and automated material handling systems.

4 CONCLUSIONS AND DISCUSSION

System-analysis integration methodologies for DELS automate generation of analysis models, such as discrete event simulation, from formal system models. Development and reuse of these methodologies can be simplified by explicitly incorporating abstraction. The system-analysis integration approach in this paper first models abstractions common to many system and analysis models in the DELS domain, rather than integrating individual simulation or analysis tools. This maximizes reusability of SAI methods, and any corresponding tools. On-going research seeks to extend and refine existing taxonomies to the level of detail required by detailed system models.

This paper presents two contributions to modeling systems and related abstractions that are the foundation for many analysis models. Generalization is used to formalize the relationship between system and abstract models. Multi-layer abstractions can be used to generate many different types of analysis models for a variety of specializations of DELS. Abstractions are captured in libraries of components that are linked formally by generalization and organized into taxonomies under the PPRF abstraction. System models are constructed from the abstractions by selecting appropriate library components from the taxonomy, specializing when necessary, and assembling them. Formalizing the relationship between system and abstract models allows abstractions to be retrieved from system models correctly and automatically.

Some analysis methods, such discrete event simulation, present many challenges to developing standard transformations between the abstract models and widely-used analysis tools and techniques. One significant challenge is the lack of complete, standard simulation model libraries for specifying the product, process, resource, and facility aspects of DELS. The lack of standard descriptions is also apparent when integrating representations of control functions, simulation components that carry out the control functions (actuator blocks), and representations of control behavior (control rules). One solution would be a standard specification, abstraction, or model library for simulation tools used in the DELS domain. Then analysis models could be constructed from the abstraction using nearly one-to-one mappings. On-going research is focused on identifying and resolving SAI capability gaps within and between system models, developing additional multi-layer abstraction models and taxonomies, and identifying the analysis model libraries they should be transformed to in simulation tools.

ACKNOWLEDGMENTS

Commercial equipment and materials might be identified to adequately specify certain procedures. In no case does such identification imply recommendation or endorsement by the U.S. National Institute of Standards and Technology, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose.

REFERENCES

Ahuja, R. K., T. L. Magnanti, and J. B. Orlin. 1993. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall.

Beck, J. C., P. Prosser, and E. Selensky. 2003. "Vehicle Routing and Job Shop Scheduling: What's the Difference?". In *ICAPS*, 267–276.

Drezner, Z., and H. W. Hamacher. 1995. *Facility Location*. Springer-Verlag New York, NY.

Estefan, J. A. 2007. "Survey of Model-based Systems Engineering (MBSE) Methodologies". *INCOSE MBSE Focus Group* 25:8.

Frantz, F. K. 1995. "A Taxonomy of Model Abstraction Techniques". In *Proceedings of the 27th Winter Simulation Conference*, edited by C. Alexopoulos, K. Kang, W. R. Lilegdon, and D. Goldsman, 1413–1420. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

ISA 2010. "ANSI/ISA-95.00.01-2010 (IEC 62264-1 Mod) Enterprise-Control System Integration - Part 1: Models and Terminology".

Lenstra, J. K., and A. R. Kan. 1975. "Some Simple Applications of the Travelling Salesman Problem". *Journal of the Operational Research Society* 26 (4): 717–733.

Marzolla, M. 2010, June14–16. "The `qnetworks` Toolbox: A Software Package for Queueing Networks Analysis". In *Analytical and Stochastic Modeling Techniques and Applications, 17th International Conference, ASMTA 2010, Cardiff, UK, Proceedings*, edited by K. Al-Begain, D. Fiems, and W. J. Knottenbelt, Volume 6148 of *Lecture Notes in Computer Science*, 102–116: Springer.

Mönch, L., P. Lendermann, L. F. McGinnis, and A. Schirrmann. 2011. "A Survey of Challenges in Modelling and Decision-making for Discrete Event Logistics Systems". *Computers in Industry* 62 (6): 557–567.

OMG 2015. "OMG Unified Modeling Language (OMG UML) Version 2.5".

Pham, D.-N., and A. Klinkert. 2008. "Surgical Case Scheduling as a Generalized Job Shop Scheduling Problem". *European Journal of Operational Research* 185 (3): 1011–1025.

Smith, S. F., and M. A. Becker. 1997. "An Ontology for Constructing Scheduling Systems". In *Working Notes of 1997 AAAI Symposium on Ontological Engineering*, 120–127.

Sprock, T. 2015. *A Metamodel of Operational Control of Discrete Event Logistics Systems (DELS)*. Ph. D. thesis, Georgia Institute of Technology, Atlanta, GA.

Thiers, G. 2014. *A Model-Based Systems Engineering Methodology to Make Engineering Analysis of Discrete-Event Logistics Systems More Cost-Accessible*. Ph. D. thesis, Georgia Institute of Technology, Atlanta, GA.

Walrand, J. 1988. *An Introduction to Queueing Networks*. Prentice Hall.

AUTHOR BIOGRAPHIES

TIMOTHY SPROCK is an Industrial Engineer in the Systems Engineering Group under the Systems Integration Division (SID) of the Engineering Lab (EL) at the National Institute of Standards and Technology (NIST) specializing in model-based systems engineering methodologies for discrete event logistics systems (DELS). He holds a PhD in Industrial Engineering from Georgia Tech. His email address is timothy.sprock@nist.gov.

CONRAD BOCK is a Computer Scientist in the Systems Engineering Group under the Systems Integration Division (SID) of the Engineering Lab (EL) at the National Institute of Standards and Technology (NIST)

specializing in formalization of engineering modeling languages. His previous experience includes business process modeling with SAP and Microsoft. Mr. Bock leads efforts on process modeling in UML and SysML at the Object Management Group. He studied at Stanford, receiving a B.S. in Physics and an M.S. in Computer Science. His email address is conrad.bock@nist.gov.