NISTIR 8326

A Model-Driven Approach to Interoperability Between Simulation and Optimization for Production and Logistics Systems

Timothy Sprock

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8326

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

NISTIR 8326

A Model-Driven Approach to Interoperability Between Simulation and Optimization for Production and Logistics Systems

Timothy Sprock
Systems Integration Division
Engineering Laboratory

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8326

December 2020

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

National Institute of Standards and Technology
Interagency or Internal Report 8326
Natl. Inst. Stand. Technol. Interag. Intern. Rep. 8326, 34 pages (December 2020)

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8326

Abstract

Simulation and optimization must be used together to design and operate large-scale, complex systems. Integrating them presents a number of conceptual and technical problems, which can be addressed with more expressive, computer-interpretable system models. This paper presents a model-driven approach to integrating simulation and optimization methods by exchanging system designs that specialize analysis abstractions, both defined in the Systems Modeling Language (SysML). The analysis abstractions enable translation of system models to corresponding analysis models and tool interfaces. This approach (model-driven system-analysis integration) is demonstrated by developing a multi-fidelity, multi-method simulation optimization methodology and applying it to a supply chain design case study.

Key words

Analysis Interoperability; Discrete Event Logistics Systems (DELS); Industrial Engineering; Model-based Systems Engineering; Production and Logistics Systems Modeling; Simulation Optimization; SysML;

Table of Contents

List of Figures

1. INTRODUCTION

Increasing system scale and complexity require improved decision-making support in Discrete Event Logistics Systems (DELS), which are networks of interconnected resources and subsystems that add value by operating on items flowing through them [1, 2]. These include supply chains, manufacturing systems, transportation networks, warehouses, and health care delivery systems. Combining simulation and optimization methods leverages their complementary strengths to provide full lifecycle decision-making support for DELS, from system design methods to online real-time operational control. Despite evidence that these methods can be used together, there does not appear to be a widely-adopted, generic way to integrate existing analysis (simulation, optimization) tools or establish more general interoperability among seemingly-related methods analyzing the same system.

One challenge to integrating simulation and optimization models, methods, and tools is that a single simulation model is often used as the *complete* system model, even though it often captures only one narrow view of the system, as needed for its kind of simulation. This limits optimization methodologies to searching only alternatives that can be expressed from the perspective of a single simulation model. Integrated simulation-optimization methodologies need access to multiple simulation models to optimize the entire (all views of) the system. This principle is essential to developing and integrating specialized, domain-specific (or purpose-specific) simulation and optimization tools and developing simulation-optimization methodologies that integrate different tools, abstractions, formalisms, etc. Multiple views, each captured as a different analysis model, assumes that there exists a separate, complete system model that can be accessed readily and inexpensively when constructing analysis models.

This paper proposes integrating simulation and optimization methods by exchanging computer-interpretable system models that are expressed in standard information formats (syntax) interpreted in standard ways (semantics), as illustrated in figure 1. For example, DELS simulation and optimization models would benefit from standard formats and interpretations for items flowing through a system (types and quantities), how they are flowing (paths and resources), and when they are flowing (control). System models describe logical relationships between various aspects of the system and its environment, as compared to analytic and geometric models. Standard modeling languages such as the Systems Modeling Language (SysML) are more expressive than analysis languages, enabling precise analysis-independent specifications that are not constrained by any target analysis language. SysML covers structure, behavior, and control of systems independent of analysis, enabling a single system model to be the source

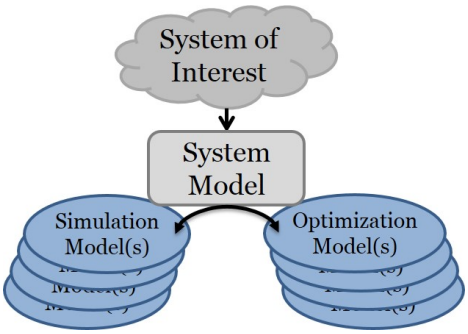

Fig. 1. Simulation and optimization models are constructed from and exchange results via a common system model.

for creating many kinds of analysis models, including purpose-specific simulation and optimization models. Exchanging system models between simulation and optimization tools enables analysis models to be generated and updated as needed to reflect required views, new solutions, etc.

The research reported here shows that system models can be linked to reusable analysis abstractions via specialization, facilitating construction of analysis models. Analysis languages, tools, and methods can be adapted to these shared abstractions, including those for specific applications. This work is an extension to the research reported in [2–4]. The discussion and examples in this paper primarily focus on discrete event simulation, however, the methods could apply to other kinds of analyses. The approach could also apply to platforms other than the one validated here and between those platforms. Further descriptions and reference implementations for these other analyses and platforms are deferred to a future paper. The SysML model libraries in this paper can be found at [5, 6], along with documentation [2]. The case study implementation can be found at [7].

The rest of the paper is organized as follows: section 2 shows some benefits of simulation and optimization interoperability via use cases in DELS analysis. Section 3 describes the (system) model-driven approach to simulation-optimization interoperability, including definition of analysis abstractions and specialization of system models from them, as well as methods for adapting existing analysis methods and tools to those abstractions, including generating analysis models from them. Section 4 summarizes a case study applying the methods detailed in the previous sections to a methodology for designing and analyzing a distribution supply chain.

2. Use Cases for DELS Analysis Interoperability

Research and applications become more specialized and reliant on interoperability of individual analysis components as systems become larger and more complex. Researchers (analysis developers) focus on their own particular area of interest (algorithms or domains) and rely on others to implement complementary analysis components. General approaches to integrating multiple analysis methods and their associated tools, such as for simulation and optimization, remains a largely unmet challenge.

Simulation optimization methods are commonly integrated by defining interfaces between simulation and optimization tools for exchanging input parameter values (input to simulation) and solution values and other feedback, such as gradient information (output to optimization), for example, in the SimOpt library [8] or Industrial Strength Compass [9]. The commercial success of OptQuest™ can be attributed partly to its integration with several popular simulation tools. However due to semantic differences between OptQuest and simulation, integrating OptQuest with each simulation platform is an expensive process that does not scale well to many-solver environments [10].

Improving interoperability between simulation and optimization methods requires standardized information exchange between the two methods, both in format (syntactic) and interpretation (semantic) [11]. Integration is currently achieved by two mappings: syntac-

tic mappings for each pair of tools (one for each pair), and semantic mappings for each problem instance (analysis model) created using a particular pair of tools. The semantic mapping is often done by "promoting" (mapping) parameters from a simulation model to variables in an optimization model. However, this approach relies on the simulation tool to interpret and implement the optimization solution. Interpreting optimization model output is often hard-coded into the simulation interface. Implementation might be as simple as setting simulation parameter values already mapped to the optimization variables, but only if optimization methods are limited to simply searching over parameter values. In complex cases such as changing system topology, implementation might require the simulation model to be rebuilt from scratch. Since the simulation tool interprets and translates both simulation and optimization input/output, simulation optimization methodologies have limited ability to incorporate multiple simulation methods and tools.

The following sections describe three broad use cases for improved simulation optimization interoperability: specializing simulation optimization for particular DELS domains (section 2.1), generating and evaluating domain-specific simulation models (section 2.2), and integrating multiple analysis methods into a simulation optimization methodology (section 2.3).

2.1 Domain-Specific Simulation Optimization Specializations for DELS

Domain-specific characteristics can be exploited by simulation optimization methods to provide narrowly-scoped and well-structured search neighborhoods. In genetic algorithms, domain-specific structures can be exploited through specialized encoding schemes, initialization procedures, and local search operators; see, for example, [12–17]. In Tabu methods, domain-specific search neighborhoods can be integrated into local search procedures, such as swapping orders within the schedule [18], separating routing and scheduling components of the job shop problem [19], and setting Kanban levels [20]. Knowledge-based optimization methods incorporate learning modules that use domain-specific information to guide the search process, such as selecting priority rules and lot sizes [21]. Specialized algorithms based on standard system models can guarantee consistent performance comparisons between competing algorithms [20, 22]. Domain-specific analysis methodologies exploit domain characteristics to enable faster, cheaper, or better solution methods than their generic counterparts. Shared system models (system conceptualizations) enable developing increasingly specialized tools that can be easily integrated with other tools developed for the same domain or abstraction, creating an ecosystem (libraries) of plug-and-play methods.

2.2 Generating and Evaluating Large-Scale Simulations

A major challenge facing simulation optimization is the expense of constructing and evaluating large-scale, high-fidelity simulations. It has long been recognized that the cost and complexity of simulation model development needs to be reduced, including in conceptual modeling methods, and that one way to do this is through model and component reuse

[23]. Simulation models can be automatically generated from libraries of reusable simulation components, model templates, and small families of flexible models [24–26]. The core manufacturing simulation data (CMSD) standard supports exchange of manufacturing information between simulation and manufacturing applications [27, 28]. Domain ontologies and domain-specific languages can narrow the gap between simulation specification model and simulation program [29, 30].

Large-scale simulation development shares many challenges with systems and software engineering methods and could benefit from their model-driven methodologies [31]. Model-driven architecture (MDA) and model-based systems engineering (MBSE) methodologies involve platform-independent models separate from their implementations in specific optimization or simulation software packages [32, 33]. MDA and MBSE methodologies use platform-independent system models to automatically generate analysis models via computer-interpretable, reusable model-to-model (M2M) translations from a single system model into target analysis languages [34–37]. Generative methods for constructing simulation models from validated model library components are effective especially when a simulation tool requires complex combinations of platform-specific simulation components to express even simple concepts [26, 38, 39]. Model-driven generative approaches use platform-independent reference models to define platform-specific simulation model library components and patterns (rules) for assembling model library components in the target language [38]. A generator-based approach is discussed further in section 3.3.

Simulation run-time efficiency challenges are also widely recognized [23, 40, 41]. Specialized formalisms, abstractions, or tools can reduce simulation execution time compared to general purpose simulation tools [42–45]. Innovations, such as domain-specific simulators, could benefit from basing method/tools on a standard system model or analysis abstraction, which would make it easier to integrate with other models.

2.3 "Multi-*X*" Analysis Methodologies

Model-driven approaches integrate multiple sources of information, often from heterogeneous systems, disparate sources, and incompatible formats, to create an complete, integrated model of the system. This system model enables generation of many kinds of analysis models or many instances of the same kind of analysis. Analysis models generated using specialized formalisms, abstractions, and tools can be combined into "multi"-method analysis methodologies. Semantic (model) interoperability enables developing simulation-optimization methodologies that are composed from predefined simulation and optimization components.

Multi-fidelity ordinal transformation methods use low-fidelity simulation models to evaluate more solutions, which then provide guidance for selecting solutions to evaluate using high-fidelity simulation models [46]. Multi-fidelity simulation methods for production and logistics systems have been studied, including when and how to to use system approximations [47–50]. Similarly, reduced simulation methods decrease overall simulation effort by reducing the resolution of non-bottleneck systems, see, for example, semiconduc-

tor wafer fabrication simulation [51].

"Multi"-method simulation optimization methodologies enable selecting and integrating appropriate simulation tools for the behavior modeling required; see discussion on cross-paradigm simulation modeling, including system dynamics, discrete event, and agent-based paradigms [52], discrete event simulation modeling paradigms [53], and a multi-formalism, multi-solution framework that integrates multiple languages and tools using a common intermediate abstraction of the domain of interest [42].

Multi-fidelity methods assume that appropriate surrogates (consistent with or derived from high-fidelity models) can be readily identified, and multiple simulations (views) can constructed inexpensively, potentially from a single system model. Our methodology proposes using object-oriented modeling and programming methods, specifically generalization and composition, to organize and retrieve system abstractions used to generate reduced or low-fidelity surrogates (section 3.2). Section 3.3 describes organizing and generating multiple kinds of analysis models from a common representation.

3. A Model-driven Approach to Integrating Simulation and Optimization

Object-oriented modeling and programming methods have been used to develop machine-readable production and logistics system models (see references in section 3.1). Formal system models can be exchanged between analysis tools and directly translated into executable analysis models. The pitfall so far has been system models and translations that are too specific to a particular kind of system or target analysis language/tool. It is easier to manage architecture and complexity this way, but limits reusability beyond the original system (domain) or selected analysis tool. The challenge is to provide models at a sufficient level of abstraction to justify developing domain-specific tools and methods (or adapting existing ones), while also being specific enough to drive productivity enhancements from these modeling and analysis tools and methods.

This section describes a model-driven approach that bridges between domain-specific terminology used by practitioners to describe their systems and common abstractions used to formulate analysis models. Traditionally, mapping and translation between the two is done using ad-hoc, time-consuming methods reliant on analysis experts. The proposed method formalizes system models and analysis abstractions separately and formally links the two by specializing the system models from the abstractions. Section 3.1 describes using object-oriented system models to define domain-specific interfaces between simulation and optimization methods and tools. Section 3.2 describes methods to link system models to the abstractions used to construct analysis models. Section 3.3 describes a system-analysis integration method that uses system models and abstractions to generate required analysis models, in this case simulation models. Model-driven integration methods utilize generation capabilities to adapt existing tools to support common abstractions and ensure that analysis models are consistent by generating (updating) from the same source.

These methods are applied to the simulation optimization case study on distribution supply chain design detailed in section 4, illustrating the proposed modeling methods.

In these systems, customers ship packages (or generic commodities) to other customers through the network. Examples of this type of supply chain include large-scale distribution companies such as FedEx or UPS or local courier services that move packages between customers (or businesses). These models can also be applied to material handling systems within warehouses or manufacturing plants that shuttle products between departments or workstations. Parts of the case study are threaded through this section to illustrate these methods.

3.1 Domain-specific Simulation-Optimization Interfaces Defined by System Models

One strategy to increase interoperability and consistency between domain-specific simulation and optimization algorithms is to develop analysis models from agreed-upon, complete definitions of the actual system of interest: a system model. Then methods/tools can be implemented to exchange information conforming to the system model or the model itself.

Some simulation-optimization methodologies exchange optimization solutions and simulation evaluations by posting system state (model) to a global blackboard or storing parameters and solutions in a common relational database [54, 55], respectively. Several authors describe methodologies where the optimization model and corresponding simulation model are generated automatically from a `System Model Object` [17, 56, 57]. McLean et al. [58] present a conceptual reference architecture for integrating distributed manufacturing simulation models, other software applications, and data repositories. Using network-based abstractions to bridge between DELS system and analysis models was introduced in [59] (section 3.1.2). Object-oriented languages/methods provide flexible and reusable frameworks for capturing supply chain descriptions [60, 61]. Some frameworks exclusively target the simulation analysis domain [26, 62–64]. Several object-oriented reference models for supply chains are reviewed in [65].

Previous research on generating analysis models from system models relied on system models and mappings/translations to analysis that are too specialized for one kind of system or analysis tool (trading reusability for simplicity). Model-driven system-analysis integration methods based on commonly-used abstractions can be reused for many domains and extended to support more specialized applications. The integration methods proposed in this paper exchange platform-independent system and abstraction models specified using SysML. OMG's SysML, an extension of UML, provides an object-oriented modeling environment used in many system engineering design methods [66]. Modeling languages like SysML support abstraction, model library construction, and modeling precision and expressiveness. SysML offers a standards-based, platform-independent approach for creating system models and integrating them with platform-specific tools, such as simulation tools [32, 33]. The next two sections describe system models and abstraction models constructed using SysML.

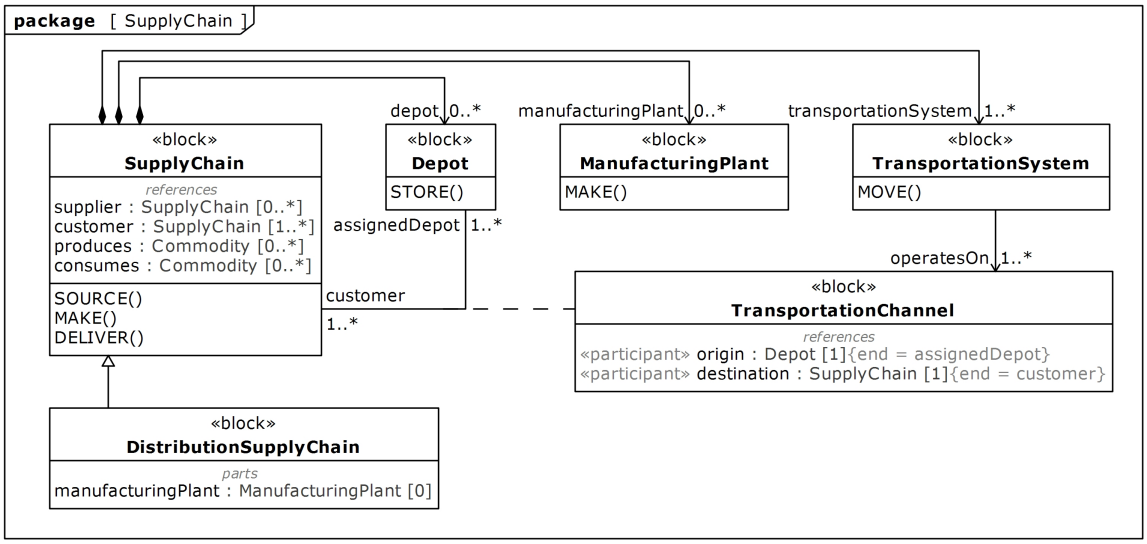

Fig. 2. Distribution supply chain system model

3.1.1 Distribution Supply Chain System Model

The distribution supply chain case study includes a system model specified in SysML, derived from a description more general supply chains, shown in figure 2. `Supply Chains` are composed of `Manufacturing Plants`, `Transportation Systems`, and `Depots`. Composition, a whole-part relationship, is shown in SysML by a black diamond, with the whole on the diamond end and the part on the other end. These components broadly align with *Make*, *Move*, and *Store* capabilities. `Distribution Supply Chains` are defined here as specialized `Supply Chains` that do not have manufacturing plants, indicated by the multiplicity *[0]* next to that property (role) of `Distribution Supply Chain`. Generalization (specialization) is represented by a hollow headed arrow pointing from a specialized class to a more abstract one.

Supply chains source items (`Commodities`) from suppliers, transform them, and deliver them to customers. In object-oriented modeling, *suppliers* and *customers* are modeled as roles (properties) played (typed) by other supply chains. These roles are referenced by the supply chain allowing interaction between the supply chain and its suppliers and customers, but denoting these entities are not owned by the supply chain. Role-based modeling allows the same supply chain (enterprise) to act as both suppliers and customers (roles) at different times with different responsibilities and interaction patterns. Suppliers and customers are modeled as `Supply Chains` to enable modeling multi-tier supply chains and flexibility for elaborating their internal details. For example, they can either be modeled as black box entities or decomposed into their respective manufacturing, storage, transportation, and sub-tier supply systems.

Supply chain components are associated via `Transportation Channels` operated on by the `Transportation System`. In distribution supply chains, customers (supply chains

playing the *customer* role) send items (`Commodities`) to each other via transportation channels and depots (used for efficiency). Each customer can be assigned to (serviced by) multiple depots. Assignment is represented by an association (typed by `TransportationChannel`) involving one supply chain and one depot (playing the *customer* and *assignedDepot* roles in the channel, respectively). `Commodities` play several roles in the supply chain. In addition to sourced and delivered items, items that supply chain *produces* and *consumes* are played by `Commodities`. As with the supplier and customer roles, these roles (properties) are also referenced, not parts. Figure 2 omits value properties, for brevity, such as key performance indicators; fixed costs for opening depots and purchasing trucks for each depot; and variable costs for trucks traversing transportation channels.

In addition to defining the kinds of supply chain components and their relationships, the system model must also capture behavior of components using behavioral models, such as activities or state machines. The `Transportation System` dispatches trucks (transportation resources) from depots to customers to drop off or pick up items (commodity instances). Trucks are modeled as uncapacitated transportation resources, and in this case study are based out of (assigned to) a particular depot. Once a truck is assigned to service a customer, it will drop off all items destined for that customer and pick up items from that customer to be routed through the depot. The same is true for transportation from depot to depot. In this case study, each depot is approximated as a cross-dock with no processing requirements and unlimited storage capacity. It simply places each incoming item into an outbound transportation queue to await a truck. Each item flows through the supply chain according to route, a pre-defined sequence of transportation channels. This flow path is constructed during the design process for each kind of commodity. Finally, a control policy determines which customer to service next when a transportation resource becomes available. Section 4.4 describes the control policy selection process.

Object-oriented system models, such as those constructed in SysML, can be extended to refine simplified abstractions, such as those described above, with more detail. For example in this case study, depots are approximated initially as cross-docks, a simplification that requires less effort specifying details that may not be used if a particular location is not selected, reduces the resolution of each depot's behavior, and speeds up early design exploration phases. However, each depot (location) selected during the design (optimization) process can be refined later with an internal behavior specification. These refinements can then be translated into more complex simulation models. Model-based systems engineering methods reuse and refine existing system and analysis components, simplifying and reducing system design and model development efforts.

3.1.2 Formal Abstract Model: Network-based Abstractions

Developing reusable model-based analysis interfaces to support system-analysis integration is challenging because system and analysis models are often specified at different levels of detail (abstraction). Linking the two requires a formal model of system abstractions used to generate analysis models. Formal abstractions for analysis specify syntax and [execution]

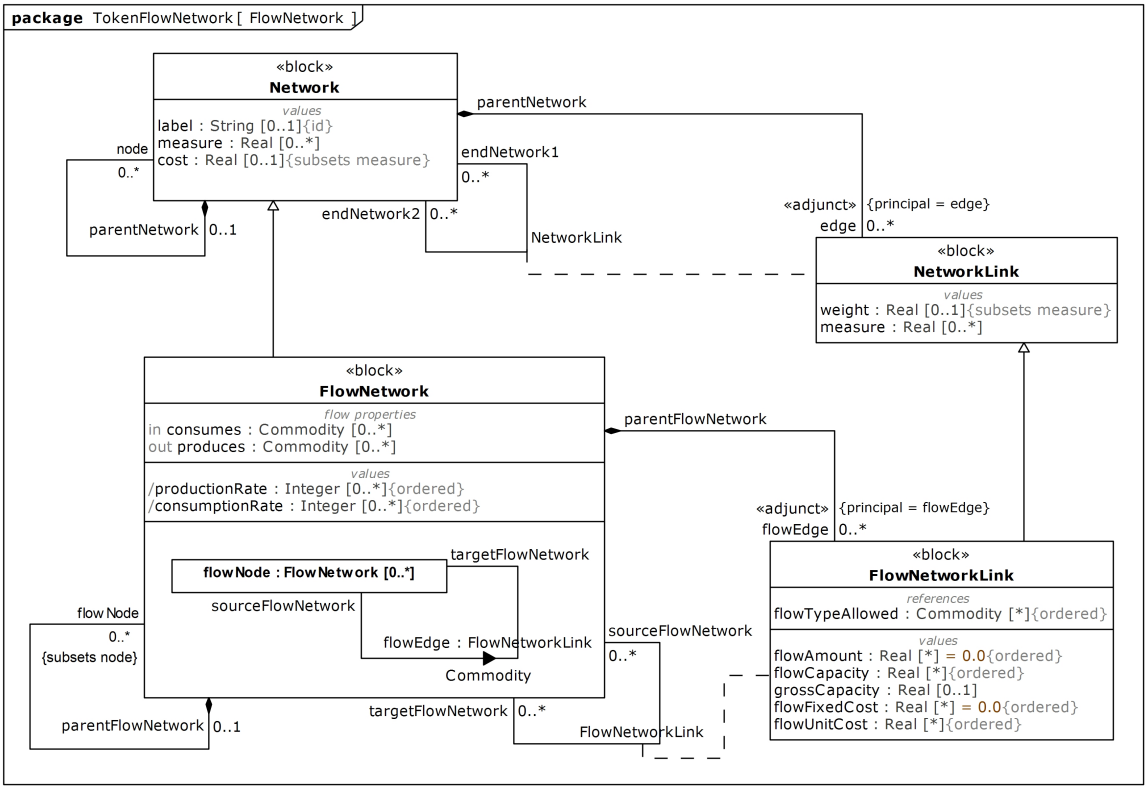

Fig. 3. Flow Networks are foundational to constructing many kinds of analysis models, including discrete event simulation.

semantics used by analysis tools.

Networks are common system abstractions used to construct system models and generate analysis, shown in figure 3. For example, the Facility Location Problem (FLP) uses the `Network` definition to formulate an optimization model for selecting storage or manufacturing facilities (*nodes*/locations) and *edges* linking supplier and customer nodes to facility nodes. Formalization of network abstractions and application to constructing analysis models was first described in [59]. The network abstractions described here were further refined in [2].

`Flow Networks`, a specialized kind of `Network`, are foundational to constructing many kinds of analysis models, including discrete event simulation (figure 3 formalizes [67]). Each `Flow Network` is composed of other flow networks playing the role of *flowNodes* (parts of a flow network). Composition, a whole-part relationship, is shown in SysML by part rectangles in graphic compartments of blocks (in addition to black diamond associations between blocks). Part-part relationships are expressed in those compartments as connectors, which are roles played by associations, between the rectangles. In flow networks, these are `flowEdges` roles played by `Flow Network Links`, between source and target flow nodes.

`Commodities` are items that flow across flow edges from source to target. The pro-

duction and consumption of commodities by a flow network are expressed as properties, *produces* / *productionRate* and *consumes* / *consumptionRate*, respectively. Each *flowEdge*, typed (played) by `FlowNetworkLink`, has several flow-related properties, such as the type of commodities allowed to flow across the edge (*flowTypeAllowed*), maximum flow rate of commodity items across the flow edge (*flowCapacity*), and cost for a commodity to traverse the edge (*flowFixedCost* & *flowUnitCost*).

3.2 Abstraction: Formulating Analysis Models from System Abstractions

Most routine analysis methods and their corresponding tools are applied to analyze abstractions of the system, such as flow networks described in the previous section. While sufficient interest is focused on analysis algorithm performance, it is mostly assumed that the model (input data) itself has been validated, or at least could be. However, constructing the correct analysis model for a system or domain is time consuming and challenging, due in part to extracting the correct abstraction of the system for the desired analysis and maintaining consistency across multiple abstractions of the same system for different analyses.

Two abstraction methods described in [68], model boundary and behavior modification, can be formalized using object-oriented languages. Model boundary modification creates taxonomies of model elements related by explicitly-stated (modeled) simplifications and assumptions. Object-oriented languages formalize this with generalization. Model behavior modification methods aggregates components and behavior into systems and aggregate behaviors, respectively. Object-oriented languages formalize this with composition. This section focuses on model boundary modification, but the composition relationship (*whole-part*) for networks (figure 3) covers model behavior modification methods.

Generalization formalizes the mapping between system models and their abstractions. It is shown in UML by a hollow headed arrow pointing from a specialized class to a more abstract one. For example, the generalization from `Supply Chain` to `Flow Network` in figure 4 specifies that every system fitting the description of `Supply Chain` will also fit the description of `Flow Network`. This formal relationship between two model elements guides construction of translations from the system model (of supply chains) to the abstraction (of flow networks). This method produces a 'correct by construction' abstraction, where the correct abstraction of the system model is extracted naturally [4].

Analysis models used when designing supply chains and other DELS commonly require abstracting the system model to (flow) network abstractions. Facility location problems (FLP) are formulated from a network definition (abstraction) of the supply chain. FLP determines the best nodes to optimize network configuration; for example, selecting new depots (locations). To extract a `Network` from a `Supply Chain` definition (figure 4), *customers* and *depots* (parts of the `Supply Chain`) are mapped (abstracted via generalization) to *nodes* and connectors typed by `Transportation Channels` are abstracted to *edges* (modeling customer-to-depot assignments). Multi-commodity flow network (MCFN) models are formulated by abstracting the `Supply Chain` to a `Flow Network`. Supply chain *customers* and *depots* are abstracted to *flowNodes* (parts of `Flow Network`). `Transport-`

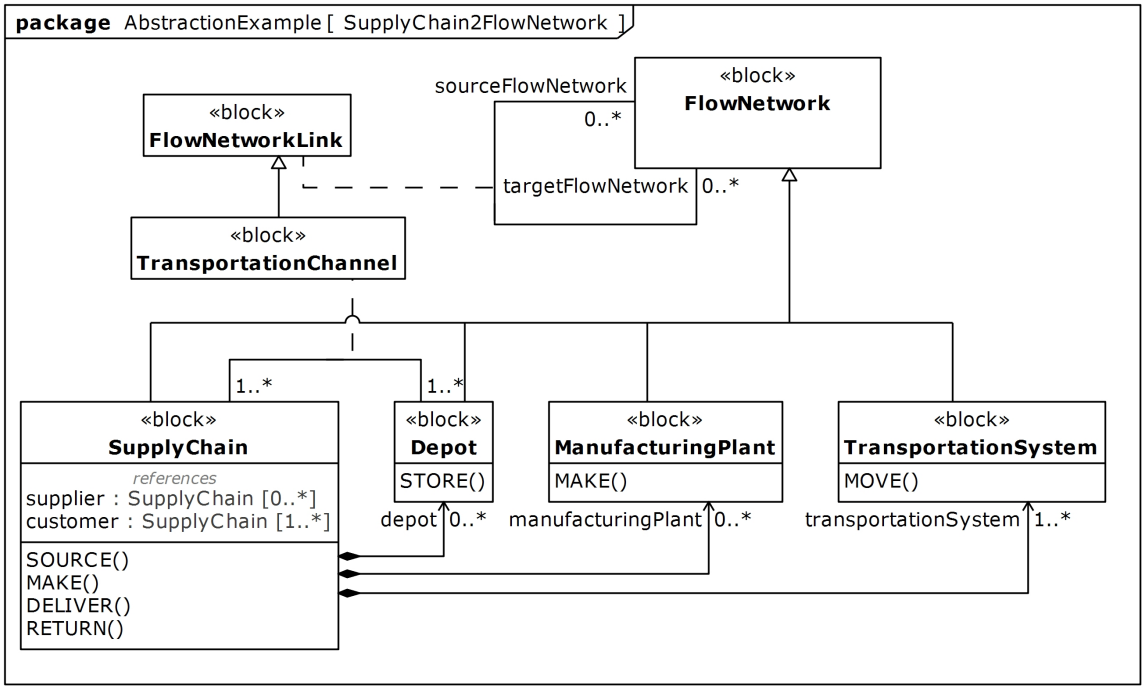

Fig. 4. Supply Chains are specialized Flow Networks. Generalization enables Flow Networks to be recovered and used for generating MCFN analyses. The distribution supply chain class is omitted from the picture for brevity, but remains a part of the model.

ation Channels are mapped to *flowEdges* (modeling flow between customers and depots).

Network optimization in the case study uses the Flow Network abstraction, solving the FLP with flow capacity constraints and optimized commodity flow paths derived from the MCFN model. Only one mapping from supply chains to flow networks is needed because Flow Network is a kind of Network (linked by generalization), and *flowNode* and *flowEdge* redefine *node* and *edge* properties, respectively (figure 3). Redefinition in SysML links properties of a specialized class to properties in a more abstract class.

The joint FLP and MCFN optimization model is formulated using properties and constraints defined by both Network and Flow Network, respectively. For example, formulating FLP analysis models requires defining the cost of serving each customer from each depot and mapping this value to Flow Network Link's *weight* value property. This value is approximated by the distance from customer to depot. These approximations can be refined as additional information becomes available. For example, *weight* can be refined to reflect the travel distance multiplied by the expected amount of goods that need to be shipped to/from that customer (expected flow) or incorporate economies of scale with piece-wise linear edge cost/capacity.

One advantage of generalization is that abstractions for constructing analysis models can be extracted automatically from detailed system models. Ideally, generalizations from

system models to abstractions are inferred automatically by constructing system models (again by generalization) from existing domain-specific model libraries and reference models that are already generalized by abstractions for analysis generation. This approach reduces errors and inconsistencies resulting from ad-hoc mappings or manual construction of an auxiliary abstraction model and reduces the time required to verify correctness of abstractions retrieved from the system model. However, existing system models can be mapped after construction (via generalization) to domain-specific model libraries and reference models or abstractions. Mapping existing system models to abstractions (either by generalization or stereotype application) is time-consuming, error-prone, and does not leverage inheritance in object-oriented languages.

3.3 System-Analysis Integration & Simulation Generation Methods

Model-driven analysis integration methods that do not rely on the simulation model as the system model require translating the system model into analysis models. Generating and updating simulation models, however, has been particularly challenging. Model-driven methods, including object-oriented system modeling methods and standardized information models, are beneficial to developing reusable and modular analysis components (section 3.1). However, prior modeling efforts typically lacked formal language implementations, limiting the extensibility and reusability of reference models and model libraries. The system-analysis integration and simulation generation methods described here use two strategies to maximize reusability of methods and tools: separating the system specification from analysis method (loose coupling) and basing the integration on system abstractions, such as flow networks, that can be reused for many kinds of systems (section 3.1.2).

The proposed system-analysis integration method uses analysis-independent representations of the system and abstractions, with platform-specific generators to particular analysis tools. This approach separates information about the system to be analyzed (data and semantics) from platform-specific knowledge about modeling in the target analysis tools. This enables system modeling without knowledge of the kind of analysis that will be performed, creating analysis agnostic, conceptual models of the system. Developing analysis generators using abstractions that apply to many kinds of systems allows the generator to reuse a significant portion of the tool-specific generation infrastructure.

This section describes a platform-specific reference implementation that leverages the abstraction models and methods described in sections 3.1 and 3.2 to allow flexible, reusable, and extensible analysis generation capabilities. The goal is a methodology for flexibly generating simulation (analysis) models in available (commercial-strength) tools (COTS) that improves upon static scripts, generic (parameterized) programming methods, and custom purpose-built tools. This reference implementation draws heavily on software patterns described in [69], automated simulation generation methods surveyed in [26], and generative programming methods in [70]. These methods enable new classes, including new kinds of things and multiple implementations of existing things, to be added dynamically by extending rather than changing the existing code. The result is a platform-specific, flow

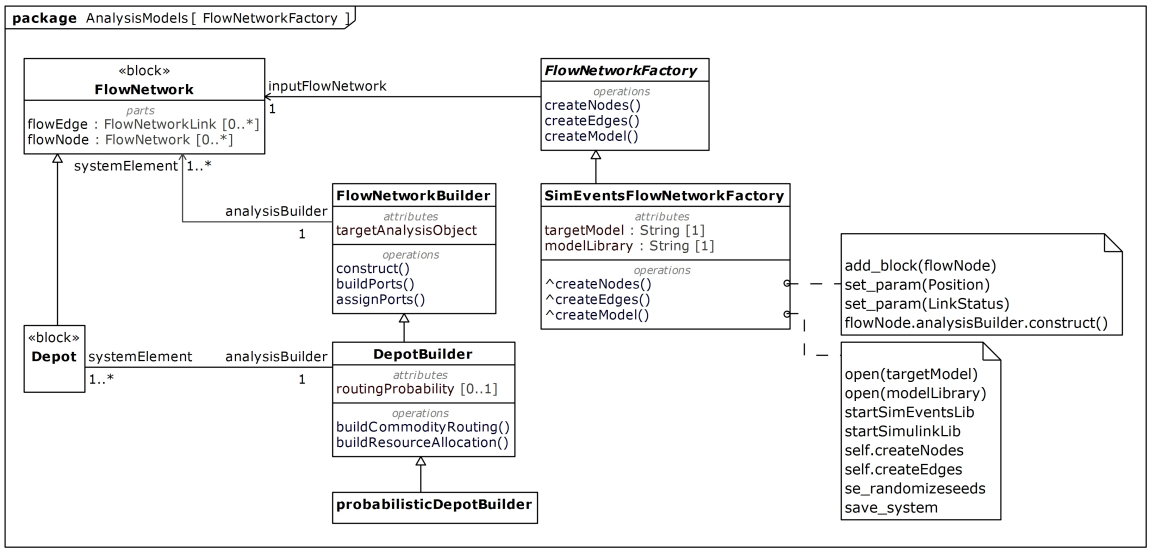

Fig. 5. Platform-specific simulation generators specialize FlowNetworkFactory. Simulation models output by the generators are all created using the same interface. Each specialized FlowNetwork has corresponding specialized analysis builders for filling in instance-specific details.

network-based discrete event simulation tool adapter specialized to support supply chains and other kinds of DELS.

3.3.1 Analysis Integration: Generative Programming Approaches to Developing Domain-specific Simulation Adapters

The `Flow Network Factory` class (figure 5) defines an interface for creating flow network-based analysis model components: flowNodes, flowEdges (connectors), and the analysis (simulation) model itself. A system model (specialized from `Flow Network`) is an input (*inputFlowNetwork*) to the `Flow Network Factory`. Figure 5 shows the relationship between platform-independent system and abstraction models (denoted by «block») and components of the platform-specific analysis generator (denoted without «block»).

Specializations of the `Flow Network Factory` interface implement its operations with tool-specific code (methods) that construct the required kind of analysis model. For example, `SimEvents Flow Network Factory` produces simulation models in Matlab's discrete event simulation software SimEvents® with pseudo-code shown in notes on the right of figure 5. This factory creates and organizes the generic (abstract) *flowNodes* of the simulation model and connects them with *flowEdges*. This step draws on generic programming techniques that leave types to be specified at a later time, usually supplied through parameters (templates). Rather than construct each complex simulation object in one pass, the implementation of *createNodes*() first copies (clones) the corresponding object from a user-supplied *modelLibrary* (described more in section 3.3.3). Then it customizes the generic simulation object into an object conforming to the (abstracted) system model.

System-analysis integration can provide access to multiple analysis methods (possibly multiple tools) by instantiating specialized factories for each kind of analysis; for example, optimization, petri nets, or queueing networks. Each factory implements the basic steps described above, first constructing a generic flow network and then customizing each object. However, each factory specialization/customization requires knowledge of the specialized system objects. In many analysis tools, analysis objects, such as simulation blocks, are not capable of self-configuration or initialization. One solution is to delegate the customization process to the system object; that is, the *createNodes*() method actually asks the input object (typed by `Flow Network`) to finish building its corresponding analysis model object. The system model has a platform-independent description of what needs to be created and could be provided with platform-specific implementation details. However, this implementation approach embeds analysis platform-specific generation code in the system object, thereby violating platform independence and potentially cluttering the object with code to support many different implementation platforms.

The solution implemented here adds a lightweight delegation mechanism on each system object. The *delegation* association (figure 5) links the system object (*systemElement*) to the analysis object (*targetAnalysisObject*) via an auxiliary builder ("helper") (*analysisBuilder* typed by `Flow Network Builder`). Then all requests to modify the simulation object (*targetAnalysisObject*) are directed to its assigned builder. For example, finishing construction (customization) of a newly created simulation object is done by *flowNode.analysisBuilder.construct*( ). These customizations may include setting parameters, building ports, customizing internal behavior, or creating metrics (data collection). For example, `Depot Builder` takes a generic depot simulation object and customizes the internal behavior by adding and configuring its resources (trucks), flow control of commodities, and scheduling customer servicing. Delegation enables `Simulation Flow Network Factory` to remain flexible by requesting the system object to complete the construction of its corresponding simulation object, but the code for doing so does not clutter the system object.

3.3.2 Specialized Builders

Multi-method environments integrate specialized tools and formalisms to provide particular analysis capabilities (section 2.3), but each analysis tool has a different way to create or specify analysis models. Specialized builder classes encapsulate code required to modify each analysis object to support a particular kind of analysis. Each class in the system model is associated with a library of builder classes that enable it to be translated into any one of many analysis tools without having to know the platform-specific details of how its behavior will be implemented in those tools. When concrete factories are instantiated, the client configures each system model object with the delegate appropriate to the kind of analysis to be created.

Delegation not only enables specialized builders to be created for each kind of analysis, but also system objects of the same kind, depots for example, to exhibit different analysis

behaviors such as varying fidelity/resolution. Different builders are required for each level of fidelity for an object. For example, `Flow Network Builder` implements low-fidelity probabilistic flow based on proportion of flow across each edge (flow network semantics). Specialized `Depot Builders` may reuse low-fidelity (probabilistic) flow routing. However, they can also implement high-fidelity control methods, such as decision rules, routing tables, or dynamic routing. These methods require more sophisticated data collection and decision-support mechanisms to be constructed in simulation blocks. The construction method is nearly identical for both the low- and high-fidelity simulation objects, except a different kind of builder is called to create the routing behavior/actuator. This approach is implemented in the case study, for example *generateLoFiDES()* (section 4.3) and *generateHiFiDES()* (section 4.4) methods (figure 7).

Specialized builders support analysis tool-specific translations by encapsulating tool-specific detail in a class that conforms/linked to the analysis independent representation. Loose coupling between system models and analysis generators enables information sources, system definition, and analysis implementation to be changed independently. Well-defined builder classes replace ad-hoc scripts that use *if/else* statements to organize specialized cases. The resulting generator is more flexible and less tied to one kind of analysis. The code to support a particular variation of an analysis model is encapsulated in the "helper" (builder) class and can be easily substituted to support different kinds of analyses. For example, an optimization flow network factory initializes the optimization model, such as an AMPL file or model object from CPLEX Application Programming Interface (API), and configures execution parameters. The *createNodes()* and *createEdges()* methods then add variables and constraints capturing production/consumption, flow balance, and flow capacity. `Flow Network` specializations, such as `Supply Chain`, customize the analysis model's behavior by adding domain-specific variables or constraints.

3.3.3 Reusable Model Libraries for Simulation Generation

Traditional MDA/MBSE approaches define and implement mappings/translations from one language to another. With the high diversity of analysis formalisms and tools (even among just discrete event simulators), expressing complex behaviors and systems produces mappings/translations that often are not one-to-one and might vary significantly from tool to tool (or even across versions of the same tool). For example, the concept of workstation may be expressed using several simulation blocks, such as combining queue, server, flow/timing/control blocks, and methods expressing different products/processes. These factors limit reusability of platform-specific translations that are based purely on language to language mappings.

The model-driven approach proposed here defines reusable mappings based on analysis-agnostic descriptions of domain-specific concepts, which stay the same even if their implementation changes across analysis platforms. These concepts form the foundation of an analysis (simulation) model library for a class of systems. Integrating a particular analysis tool requires constructing analysis library objects corresponding to standard analysis-

independent model libraries. Construction, testing, and validation of reusable analysis objects is done in the analysis tool environment. Creating custom analysis library objects that conform to standard system model libraries mitigates some differences between analysis tool languages (and their non-standard stock palettes), but requires knowledge of both the analysis tool and system library. Practically, it's easier to develop analysis library objects correctly by leveraging an analysis expert's ability to determine the best way to express each system library concept within an analysis tool's constraints (language, stock palette). This method creates a transparent, reusable, and extendable way to generate analysis objects that might only be an approximate the system object (same interface, different internals). This approach is novel because the mapping and translation from system model to analysis model is done by focusing on system and analysis concepts, rather than languages.

Flow networks are essential abstractions for constructing both system and analysis models, which makes them a good foundation for domain-specific system-analysis integration methods. The implementation described in section 3.3.1 (`Flow Network Factory`) defines automatic construction of flow network-based simulation models in a particular tool. `Flow Network Factory` treats each input as a `Flow Network`. Then specialized builders customize each simulation object according to the source DELS by adding resources or configuring control behaviors. This simulation generator can be extended by defining both a new simulation model library object and corresponding builder (*helper*) class responsible for configuring the generic simulation objects.

3.4 What is the best abstraction: A Formal Domain Model for Discrete Event Logistics Systems (DELS)

Previous subsections describe methods to 1) formally link (via generalization) system models to abstractions used to construct analysis models, 2) separate system/abstraction models from implementations of corresponding analysis model generators, and 3) reuse these generators for any system conforming to the abstraction, i.e. that can be modeled as a flow network. The reusability of the analysis generators is based on the reusability of system abstractions. Maximizing the applicability of analysis generators requires identifying additional system abstractions that can be applied broadly to constructing system models and defining analysis integrations. This section proposes an abstraction for discrete event logistics systems (`DELS`) (figure 6), which extends `Flow Network` (section 3.1.2) to include commonalities between supply chains, production, storage, and transportation systems. Then `Supply Chain` and its components (section 3.1.1) extend the DELS abstraction, rather than flow networks (figure 4 in section 3.2). Due to space constraints, this section only sketches DELS, highlighting elements used in the analysis methodology case study (section 4). The DELS models are described in more detail in [2].

Additional modeling is needed in `DELS` to describe what is being transformed (`product`), how the product is transformed (`process`), how the transformation is executed (`resource`), and how resources are configured (`facility`). The product-process-resource-facility (PPRF) abstraction is common across manufacturing reference models, including OZONE, MA-

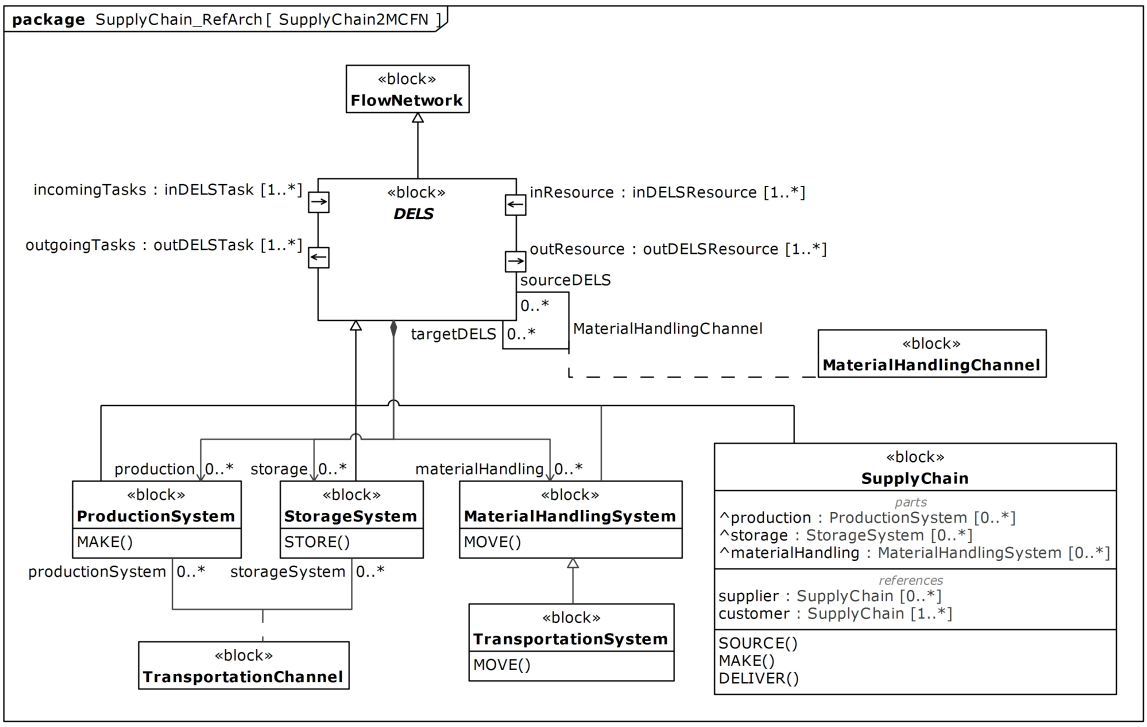

Fig. 6. Generalized supply chain models can be extended from a discrete event logistics systems (DELS) abstraction.

SON, MANDATE, and CMSD [27, 71–73]. In addition to modeling the plant's behavior, the DELS model also includes an operational control model. Operational control behaviors determine the flow of resources and tasks through the system and which resources execute those flows. Operational control is captured as extensions of processes and resources in the DELS reference model. These map decision variables in a controller's decision problem (optimization) to a particular execution mechanism (actuator) in the plant [74].

A multi-layered reference model extracts commonalities among systems and analyses at a high level of abstraction. A multi-abstraction model library allows system models to be mapped and transformed to various abstractions used by analysis models. For example, different abstractions are used to construct MCFN models (`Flow Networks`) and resource investment, sizing, and configuration models (`DELS` *resource* definitions). The DELS abstraction can be further refined into domain-specific libraries, such as production, storage, and material handling libraries (figure 6).

4. Applying Model-Driven Integration Methods to Analysis Methodologies: A Case Study

The formal (abstract) system models described in section 3.1 are used to construct and then connect analysis methods and tools for DELS. When simulation optimization methods and tools operate and interact with data conforming the system model (information model/schema), then any conforming system model (that can be mapped or abstracted) can be input to those methods. One challenge of this approach is designing tools that specifically pass information defined by the system model, rather than semantics-free data structures, such as numerical arrays. Existing analysis tools that do not do this can be adapted with interfaces that do. This case study describes an implementation of this idea. The implementation can be found at [7] under *Use Cases/DistributionNetwork*.

The case study illustrates hierarchical design of distribution supply chains where design decisions are well-structured and made sequentially using multi-stage, multi-criteria simulation optimization. It provides a rich example to demonstrate potential applications and benefits of system models mediating between multi-paradigm simulation and optimization tools (interoperability). The system model is the source for automatically generating simulation and optimization models at the required level of detail for each stage of the search process. Generative methods reduce the time and cost of using simulation optimization methods and enables simulation optimization analysis to consider structural changes to the system.

4.1 Analysis Method: Multi-fidelity Simulation Optimization

The simulation optimization methodology has three stages (figure 7): 1) *Network Design*, based on a deterministic, integer programming model formulated from a multi-commodity flow network approximation (section 4.2); 2) *Resource Investment*, a problem solved using a genetic algorithm operating on a low-fidelity simulation model (section 4.3); and 3)

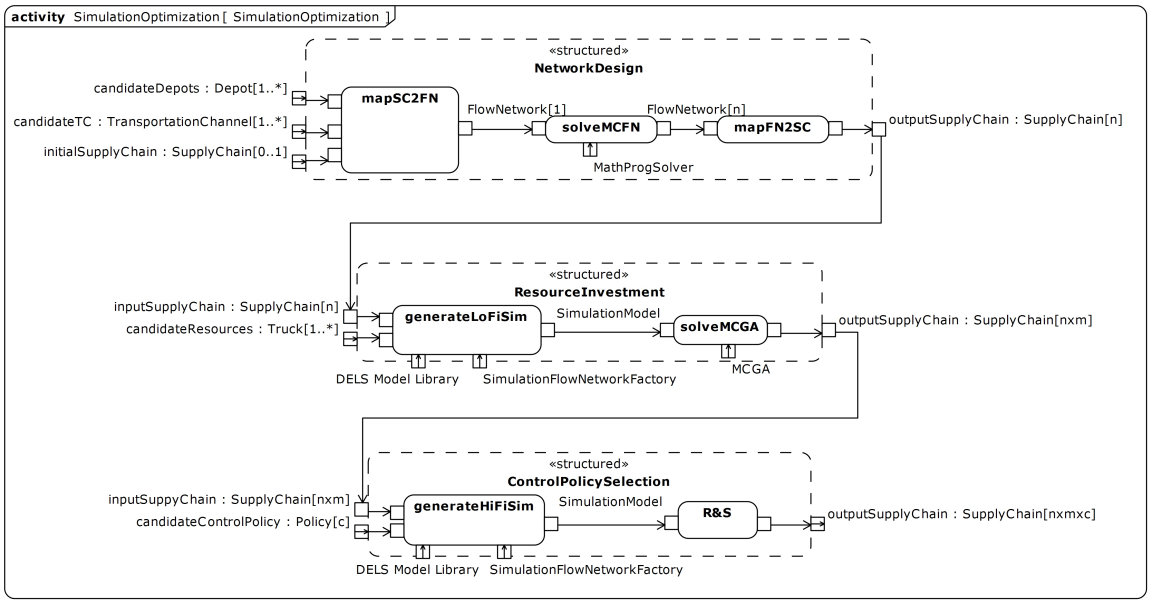

Fig. 7. An overview of the simulation optimization analysis methodology

Control Policy Selection, enumerating resource dispatch policies evaluated using higher-fidelity simulation models (section 4.4). This design process outputs a Pareto portfolio of candidate solutions (section 4.5).

The analysis stages interact by passing `Supply Chain` objects described in section 3.1. At each stage, the associated optimization and simulation models are generated at the desired level of abstraction and fidelity from the `Supply Chain` system model. The case study illustrates integrating heterogeneous methods/tools into a methodology, rather than focusing on the particular solution method implemented or the solution quality. The potential use cases (section 2) highlight that interoperable analysis methods can be incorporated to improve solution quality or run-time. This can lead to new analysis steps that further refine solutions or other aspects of the system not addressed here.

The design case study is defined as follows: the designer must select depots from predetermined candidates, specify the truck fleet size for each depot, and provide a control policy that selects the next customer to be served by an available truck. The specific problem instance in this case study was created by randomly generating 50 customers on a 240x240 grid of locations; 25 candidate depot locations on the central 120x120 grid; customer to customer annual commodity demands from a UNIFORM(0,1000) distribution. Depot selection costs are 10e4 and transportation resource fixed costs are 10e2. As a side note, the parameters were selected to produce many candidate design alternatives, as a useful illustration, rather than a single dominant depot location.

4.2 Distribution Network Design Based on a MCFN Approximation

The network design stage maps the distribution supply chain model (via abstraction / generalization) to the flow network abstraction, then formulates a combined facility location and multi-commodity flow network (MCFN) analysis model (section 3.2). This analysis model outputs depot selection, customer assignment, and flow determination (routes). The first stage exploits commercial (and open-source) math programming solvers to generate feasible, diverse, and good solutions to jump-start the local optimization process. This approach is useful for screening a large set of alternatives to quickly discard inferior solutions. This produces good-quality candidate network structures for common problems such as supplier selection or facility location [75].

By approximating these systems as flow networks (section 3.1.2), analysis tools written using `Flow Network` semantics can be applied to the network design step. In this use case, we assume the analysis component accepts a *Flow Network* (output from the *SC2FN* step), formulates a MCFN model, and solves it using a commercial off the shelf math programming solver (*MathProgSolver*). The `Flow Network` defines an interface to the analysis component formulating the MCFN optimization model. Math programming solvers will solve "any" math programming problem, i.e. its interface is defined by an optimization metamodel. Their APIs define an optimization problem class, which is instantiated with details of specific optimization models (MCFN model).

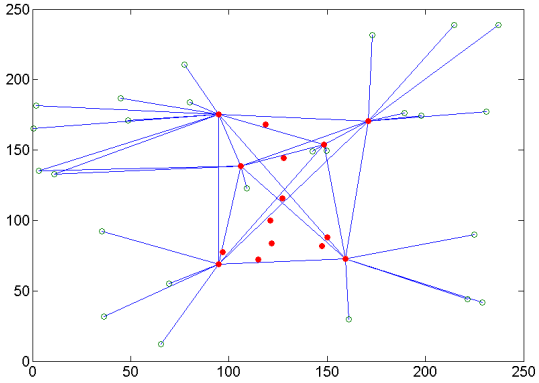

Fig. 8. Multi-commodity flow network optimization results with candidate depots highlighted in solid (red) circles

The *Network Design* stage returns a valid `Supply Chain` instance by mapping optimization analysis results from the `Flow Network` abstraction into supply chain semantics (*mapFN2SC* in figure 7). The optimal solution is displayed in figure 8. To create a pool of good candidate solutions, the facility location problem is exploited using a 'leave one out' heuristic that re-solves the problem iteratively excluding one depot selected from the optimal FLP/MCFN solution. In this case study, the initial FLP/MCFN solution selected six depots. The *Network Design* stage passes seven candidate `Supply Chain` configurations (instances) to the next stage (resource investment).

4.3 Resource Investment via Genetic Algorithm Using Low-Fidelity Discrete Event Simulation

The next design stage is the resource investment decision. Whereas MCFN analysis is a deterministic approximation of system behavior, simulation is more adept at probabilistic

modeling of system behaviors, interactions, and variability.

Because of the approximate nature of the results from the first stage (network design) and the simplicity of the surrogate control model, it seems reasonable for the next analysis stage to utilize a low-fidelity simulation model that aggregates and routes flow through the network probabilistically, rather than routing each individual item. Probabilistic flow models are reasonable approximations of control behaviors in high-fidelity simulations, but reduce overall computational expense of searching the design space to construct a portfolio of resource investment solutions. The simulation model implements a basic resource dispatch control policy that continuously allocates trucks to pick-up and drop-off routes in a round-robin manner. These two control functions will be refined, in both logic and execution, in the third stage (control policy selection).

Distribution supply chain instances (objects) created in the network design stage are used to automatically generate discrete event simulation models in SimEvents (figure 9a) using the method described in section 3.3. For brevity, the simulation model in figure 9a is a smaller instance generated from the same stock library components as the complete model used to generate the case study results. Each simulation model is embedded within a multi-objective genetic algorithm from the MATLAB® Global Optimization Toolbox™. The multi-objective genetic algorithm in MATLAB uses a controlled elitist GA variant of NSGA-II [76] to produce a Pareto portfolio of resource investment decisions. The candidate solutions are Pareto efficient with respect to resource investment capital costs (how many trucks to purchase for each depot) and achieved Service Level (defined as percentage (%) of items delivered in under 24 hours). The result is a Pareto set of 587 solutions that can be passed to the next design stage. Figure 9b displays the output of the multi-objective genetic algorithm after it operates on one of the seven system models input to this stage of analysis.

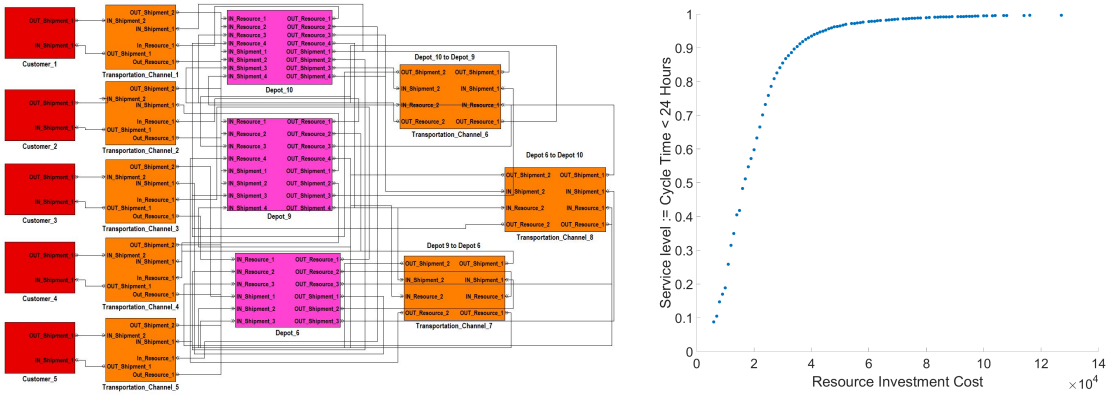

Fig. 9. (a) Discrete event simulation generated in SimEvents and (b) output of multi-objective genetic algorithm (Right)

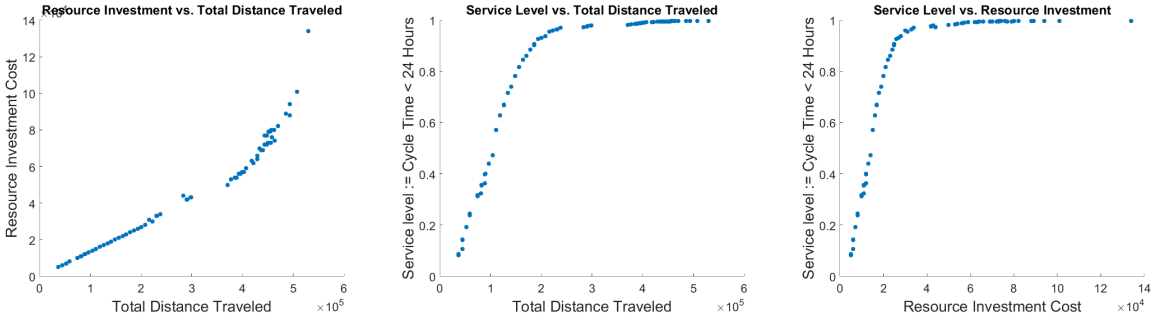

Fig. 10. Biobjective plots of the set of Pareto efficient solutions produced by the multi-objective simulation optimization.

4.4 Control Policy Selection via Enumeration Using High-Resolution Simulation

One of the outstanding challenges in design is control mechanisms. The third stage of the case study generates high-fidelity simulation models capable of evaluating a small collection of control policies that select the next customer to be serviced and dispatch a truck to drop-off and pick-up items from that customer. This stage of the analysis enumerates three control policy options with each candidate system design, though enumeration could be replaced with ranking and selection methods.

The most basic scenario uses a *round-robin policy*, which is easy to implement due to its minimal information requirements, but risks under-utilizing the capacity of each truck. The second scenario uses a *longest queue policy*, which requires the controller to gather and evaluate queue lengths from each customers. For this strategy, trucks are dispatched to the next customer as soon as they complete their previous task, which still risks under-utilizing each truck. Finally, the third scenario extends the *longest queue* policy with a *minimum queue length* before dispatching a truck to a particular customer. This improves the utilization of each truck and reduces overall distance traveled, but may increase the cycle time for shipments. This stage enumerates the complete set of control policies for each of the resource investment solutions from the previous stage. The resulting Pareto set of solutions contains 79 candidate system configurations.

4.5 Results

This multi-stage, multi-criteria simulation optimization method outputs a Pareto set of solutions. These solutions can be refined further using ranking and selection methods. For this case study, the output is a trade-off curve between transportation resource investment, total distance traveled to service customers, and the service level (cycle time satisfaction). The set of 79 Pareto efficient solutions is projected into three biobjective plots in figure 10.

5. Conclusions and Future Work

To meet analysis requirements for next-generation discrete event logistics systems, simulation optimization methods must handle the scale, complexity, and uncertainty inherent in these systems. Integration and interoperability between individual simulation and optimization methods is a key enabler to meeting those requirements. We propose integration methods driven by complete system definitions (models) for defining robust interfaces and adapters between application-specific simulation and optimization methods. Generating simulation and optimization models from the same system model improves consistency between these two complementary stages of system design, which capture different views of system structure, behavior, and control. Multiple simulations can be generated at appropriate levels of detail mirroring the diversity of available optimization methods. Simulation models can be generated in multiple languages and commercial tools, enabling high-performance simulation and spurring innovative applications of research ideas in DELS and other kinds of systems (see section 2.2).

As a case study, we construct a distribution supply chain system model by specializing flow network and DELS models, enabling multiple generic analysis models and tools to be generated from the same system specification. This suggests that *any* system conforming to the (reference) flow network and DELS models can take advantage of *any* analysis model that can be mapped to the those reference model. These reference models serve as a bridge between system and analysis models, reducing the effort of creating new analyses for existing systems and applying existing analyses to new system instances. This capability is essential for simulation optimization to be routinely used in DELS analysis.

Implementing the modeling methods for this case study required working-around some platform-specific aspects of Matlab and SimEvents, and exploiting others, for representing, storing, and exchanging system models and constructing the adapters. Future work to achieve cross-platform interoperability between simulation and optimization tools will address exchanging system models stored in a platform-independent format, such as OMG's XML Metadata Interchange standard [77]. The adapter method in section 3.3.1 demonstrates that COTS tools can expose a flow network-compliant interface. The method needs to be validated on other platforms, which may implement it differently depending on their languages and constraints.

Acknowledgments

The author thanks Conrad Bock at NIST and Leon McGinnis and George Thiers at Georgia Tech for their helpful conversations and comments.

References

[1] Mönch L, Lendermann P, McGinnis LF, Schirrmann A (2011) A survey of challenges in modeling and decision-making for discrete event logistics systems. *Computers in Industry* 62(6):557–567.

[2] Sprock T, Thiers G, McGinnis LF, Bock C (2020) Theory of Discrete Event Logistics Systems (DELS) Specification. National Institute of Standards and Technology, NIST Interagency/Internal Report (NISTIR) 8262. URL https://doi.org/10.6028/NIST.IR.8262.

[3] Sprock T, McGinnis LF (2015) A simulation optimization framework for discrete event logistics systems (DELS). *Proceedings of the 2015 Winter Simulation Conference* (IEEE), pp 2776–2787.

[4] Sprock T, Bock C (2017) Incorporating abstraction methods into system-analysis integration methodology for discrete event logistics systems. *Proceedings of the 2017 Winter Simulation Conference* (IEEE Press), pp 966–976.

[5] Sprock T, Bock C (2020) Model libraries supporting discrete event logistics systems (dels) models. *NIST Journal of Research* URL https://doi.org/10.6028/jres.125.023.

[6] Sprock T GitHub\USNISTGov\discrete event logistics systems, . URL https://doi.org/10.18434/M32203.

[7] Sprock T GitHub\USNISTGov\DELS Analysis Integration, . URL https://github.com/usnistgov/dels-analysis-integration.

[8] Pasupathy R, Henderson SG (2011) SimOpt: A library of simulation optimization problems. *Proceedings of the 2011 Winter Simulation Conference* (IEEE), pp 4075–4085.

[9] Xu J, Nelson BL, Hong J (2010) Industrial strength COMPASS: A comprehensive algorithm and software for optimization via simulation. *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 20(1):1–29.

[10] Fu MC, et al. (2014) Simulation Optimization: A Panel on the State of the Art in Research and Practice. *Proceedings of the 2014 Winter Simulation Conference* (IEEE Press, Piscataway, NJ, USA), pp 3696–3706.

[11] Tolk A, Muguira JA (2003) The levels of conceptual interoperability model. *Proceedings of the 2003 Fall Simulation Interoperability Workshop*, Vol. 7 Vol. 7, pp 1–11.

[12] Dengiz B, Altiparmak F, Smith AE (1997) Local search genetic algorithm for optimal design of reliable networks. *IEEE Transactions on Evolutionary Computation* 1(3):179–188.

[13] Azadivar F (1999) Simulation optimization methodologies. *Proceedings of the 1999 Winter Simulation Conference*, eds Farrington PA, Nembhard HB, Sturrock DT, Evans GW (ACM, New York, NY, USA), pp 93–100.

[14] Zhou G, Min H, Gen M (2002) The balanced allocation of customers to multiple distribution centers in the supply chain network: A genetic algorithm approach. *Computers & Industrial Engineering* 43(1):251–261.

[15] Syarif A, Yun Y, Gen M (2002) Study on multi-stage logistic chain network: A span-

ning tree-based genetic algorithm approach. *Computers & Industrial Engineering* 43(1):299–314.

[16] Ding H, Benyoucef L, Xie X (2009) Stochastic multi-objective production-distribution network design using simulation-based optimization. *International Journal of Production Research* 47(2):479–505.

[17] Costa A, Celano G, Fichera S, Trovato E (2010) A new efficient encoding/decoding procedure for the design of a supply chain network with genetic algorithms. *Computers & Industrial Engineering* 59(4):986–999.

[18] Yang T, Kuo Y, Chang I (2004) Tabu-search simulation optimization approach for flow-shop scheduling with multiple processors - a case study. *International Journal of Production Research* 42(19):4015–4030.

[19] Brandimarte P (1993) Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations research* 41(3):157–183.

[20] Alabas C, Altiparmak F, Dengiz B (2002) A comparison of the performance of artificial intelligence techniques for optimizing the number of kanbans. *Journal of the Operational Research Society* :907–914.

[21] Huyet A, Paris J (2004) Synergy between evolutionary optimization and induction graphs learning for simulated manufacturing systems. *International Journal of Production Research* 42(20):4295–4313.

[22] Dengiz B, Alabas C (2000) Simulation optimization using tabu search. *Proceedings of the 2000 Winter Simulation Conference* (Society for Computer Simulation International), pp 805–810.

[23] Robinson S (2005) Discrete-event simulation: From the pioneers to the present, what next? *Journal of the Operational Research Society* 56(6):619–629.

[24] Mackulak GT, Lawrence FP, Colvin T (1998) Effective simulation model reuse: a case study for amhs modeling. *Proceedings of the 1998 Winter Simulation Conference* (IEEE Computer Society Press), pp 979–984.

[25] Son YJ, Jones AT, Wysk RA (2000) Automatic generation of simulation models from neutral libraries: an example. *Proceedings of the 2000 Winter Simulation Conference*, Vol. 2 (IEEE) Vol. 2, pp 1558–1567.

[26] Cope D, Fayez MS, Mollaghasemi M, Kaylani A (2007) Supply chain simulation modeling made easy: an innovative approach. *Proceedings of the 2007 Winter Simulation Conference* (IEEE), pp 1887–1896.

[27] Lee YTT, Riddick FH, Johansson BJI (2011) Core Manufacturing Simulation Data - a manufacturing simulation integration standard: Overview and case studies. *International Journal of Computer Integrated Manufacturing* 24(8):689–709.

[28] Bergmann S, Stelzer S, Straßburger S (2011) Initialization of simulation models using cmsd. *Proceedings of the 2011 Winter Simulation Conference* (IEEE), pp 2228–2239.

[29] Silver GA, Hassan OAH, Miller JA (2007) From domain ontologies to modeling ontologies to executable simulation models. *Proceedings of the 39th conference on Winter simulation: 40 years! The best is yet to come* (IEEE Press), pp 1108–1117.

[30] Miller J, Han J, Hybinette M, et al. (2010) Using domain specific language for mod-

eling and simulation: Scalation as a case study. *Proceedings of the 2010 Winter Simulation Conference* (IEEE), pp 741–752.

[31] Gianni D, D'Ambrogio A, Tolk A (2014) *Modeling and Simulation-Based Systems Engineering Handbook* (CRC Press), .

[32] Mellor SJ, Scott K, Uhl A, Weise D (2004) *MDA distilled: principles of model-driven architecture* (Addison-Wesley Professional), .

[33] Estefan JA (2007) Survey of model-based systems engineering (mbse) methodologies. *Incose MBSE Focus Group* 25:8.

[34] Schönherr O, Pappert FS, Rose O (2013) Domain specific simulation modeling with sysml and model-to-model transformation for discrete processes. *Formal Languages for Computer Simulation: Transdisciplinary Models and Applications* :267–304.

[35] Kapos GD, Dalakas V, Nikolaidou M, Anagnostopoulos D (2014) An integrated framework for automated simulation of sysml models using devs. *Simulation* 90(6):717–744.

[36] Batarseh O, Huang E, McGinnis LF (2015) Capturing simulation tool and application domain knowledge for automating simulation model creation. *Journal of Simulation* 9(1):1–15.

[37] Thiers G, Sprock T, McGinnis L, Graunke A, Christian M (2016) Automated production system simulations using commercial off-the-shelf simulation tools. *Proceedings of the 2016 Winter Simulation Conference* (IEEE Press), pp 1036–1047.

[38] Sprock T, McGinnis LF (2014) Simulation Model Generation of Discrete Event Logistics Systems (DELS) Using Software Patterns. *Proceedings of the 2014 Winter Simulation Conference* (IEEE Press), pp 2714–2725.

[39] Huang Y, Verbraeck A, Seck M (2016) Graph transformation based simulation model generation. *Journal of Simulation* 10(4):283–309.

[40] Fowler JW, Rose O (2004) Grand challenges in modeling and simulation of complex manufacturing systems. *Simulation* 80(9):469–476.

[41] Taylor SJ, et al. (2012) Panel on grand challenges for modeling and simulation. *Proceedings of the 2012 Winter Simulation Conference* (Winter Simulation Conference), p 232.

[42] Deavours DD, et al. (2002) The mobius framework and its implementation. *IEEE Transactions on Software Engineering* 28(10):956–969.

[43] Schruben LW, Roeder TM (2003) Fast simulations of large-scale highly congested systems. *Simulation* 79(3):115–125.

[44] Angelidis E, Bohn D, Rose O (2013) A simulation tool for complex assembly lines with multi-skilled resources. *Proceedings of the 2013 Winter Simulation Conference (WSC)* (IEEE), pp 2577–2586.

[45] Mayer T, Uhlig T, Rose O (2016) An open-source discrete event simulator for rich vehicle routing problems. *Intelligent Transportation Systems (ITSC), 2016 IEEE 19th International Conference on* (IEEE), pp 1305–1310.

[46] Xu J, et al. (2016) MO2TOS: Multi-fidelity optimization with ordinal transformation and optimal sampling. *Asia-Pacific Journal of Operational Research* 33.

[47] Zülch G, Fischer J, Jonsson U (2000) An integrated object model for activity network based simulation. *Proceedings of the 32nd conference on Winter simulation* (Society for Computer Simulation International), pp 371–380.

[48] Madan M, Son YJ, Cho H, Kulvatunyou B (2005) Determination of efficient simulation model fidelity for flexible manufacturing systems. *International Journal of Computer Integrated Manufacturing* 18(2-3):236–250.

[49] Celik N, Lee S, Vasudevan K, Son YJ (2010) Dddas-based multi-fidelity simulation framework for supply chain systems. *IIE Transactions* 42(5):325–341.

[50] Chan F, Chaube A, Mohan V, Arora V, Tiwari M (2010) Operation allocation in automated manufacturing system using ga-based approach with multifidelity models. *Robotics and Computer-Integrated Manufacturing* 26(5):526–534.

[51] Hung YF, Leachman R (1999) Reduced simulation models of wafer fabrication facilities. *International Journal of Production Research* 37(12):2685–2701.

[52] Heath SK, Buss A, Brailsford SC, Macal CM (2011) Cross-paradigm simulation modeling: challenges and successes. *Proceedings of the 2011 Winter Simulation Conference* (IEEE), pp 2788–2802.

[53] Miller J, Baramidze GT, Sheth AP, Fishwick P (2004) Investigating ontologies for simulation modeling. *Proceedings of the 37th Annual Simulation Symposium (ANSS'04)* (IEEE), pp 55–63.

[54] Almeder C, Preusser M, Hartl RF (2008) Simulation and optimization of supply chains: Alternative or complementary approaches? *OR Spectrum* 31(1):95–119.

[55] Duvivier D, Dhaevers V, Bachelet V, Artiba A (2003) Integrating simulation and optimization of manufacturing systems. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* 33(2):186–192.

[56] Azadivar F, Tompkins G (1999) Simulation optimization with qualitative variables and structural model changes: A genetic algorithm approach. *European Journal of Operational Research* 113(1):169–182.

[57] Hamm M, Szczesny K, Nguyen V, König M (2011) Optimization of construction schedules with discrete-event simulation using an optimization framework. *Proc. of the 2011 ASCE International Workshop on Computing in Civil Engineering, Miami, FL, USA*, pp 682–689.

[58] McLean C, Riddick F (2000) Simulation in the international IMS MISSION project: The IMS MISSION architecture for distributed manufacturing simulation. *Proceedings of the 2000 Winter Simulation Conference* (Society for Computer Simulation International), pp 1539–1548.

[59] 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.

[60] Biswas S, Narahari Y (2004) Object oriented modeling and decision support for supply chains. *European Journal of Operational Research* 153(3):704–726.

[61] Kim J, Rogers K (2005) An object-oriented approach for building a flexible supply chain model. *International Journal of Physical Distribution & Logistics Management*

35(7):481–502.

[62] Jain S, Workman RW, Collins LM, Ervin EC, Lathrop AP (2001) Supply chain applications II: Development of a high-level supply chain simulation model. *Proceedings of the 2001 Winter Simulation Conference*, eds Peters BA, Smith JS, Medeiros DJ, Rohrer MW (IEEE Computer Society, Washington, DC, USA), pp 1129–1137.

[63] Chatfield DC, Harrison TP, Hayya JC (2006) Sisco: An object-oriented supply chain simulation system. *Decision Support Systems* 42(1):422–434.

[64] Rossetti M, Miman M, Varghese V (2008) An object-oriented framework for simulating supply systems. *Journal of Simulation* 2(2):103–116.

[65] Grubic T, Fan IS (2009) Integrating Process and Ontology for Supply Chain Modeling. *Interoperability for Enterprise Software and Applications China, 2009. IESA'09. International Conference on*, pp 228–235.

[66] OMG (2017) Omg systems modeling language (omg sysml) version 1.5, . URL http://www.omg.org/spec/SysML/1.5/.

[67] Ahuja RK, Magnanti TL, Orlin JB (1993) *Network flows: theory, algorithms, and applications* (Prentice Hall), .

[68] Frantz FK (1995) A taxonomy of model abstraction techniques. *Proceedings of the 27th Winter Simulation Conference* (IEEE Computer Society), pp 1413–1420.

[69] Gamma E, Helm R, Johnson R, Vlissides J (1994) *Design patterns: elements of reusable object-oriented software* (Pearson Education), .

[70] Czarnecki K, Eisenecker UW, Czarnecki K (2000) *Generative programming: methods, tools, and applications*. Vol. 16 (Addison Wesley Reading), .

[71] Smith SF, Becker MA (1997) An ontology for constructing scheduling systems. *Working Notes of 1997 AAAI Symposium on Ontological Engineering*, pp 120–127.

[72] Lemaignan S, Siadat A, Dantan JY, Semenenko A (2006) Mason: A proposal for an ontology of manufacturing domain. *IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications* (IEEE), pp 195–200.

[73] Cutting-Decelle AF, et al. (2007) ISO 15531 MANDATE: a product-process-resource based approach for managing modularity in production management. *Concurrent Engineering* 15(2):217–235.

[74] Sprock T (2018) Patterns for modeling operational control of discrete event logistics systems (dels). *Disciplinary Convergence in Systems Engineering Research* (Springer), pp 875–884.

[75] Osman IH (1995) Heuristics for the generalised assignment problem: Simulated annealing and tabu search approaches. *Operations-Research-Spektrum* 17(4):211–225.

[76] Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6(2):182–197.

[77] OMG (2005) Omg xml metadata interchange (omg xmi) version 2.1, . URL https://www.omg.org/spec/XMI/2.1/.